Universität Hamburg
Fachbereich Informatik

Diplomarbeit

Entwicklung von Client/Server-Anwendungen mit 4GL-Technologie

Tycoon, NATURAL und ABAP/4 im Vergleich

Betreuer:
Prof. Dr. Joachim W. Schmidt
Dr. Heiner Seibt

vorgelegt von:
Nastaran Vaziri Pour
Ottenser Hauptstr. 12
22765 Hamburg
Mat.-Nr.: 4293460

Hamburg, den 12. September 1997

Danksagung
An dieser Stelle möchte ich Herrn Prof. Dr. Joachim W. Schmidt und Herrn Dr. Heiner Seibt dafür danken, daß sie mir die Möglichkeit gegeben haben, dieses besonders interessante Thema zu bearbeiten. Herrn Manfred Gärtner, Projektleiter Bereich Lehre und Forschung bei der Software AG, danke ich für seine stete Hilfsbereitschaft in allen Fragen zu NATURAL. Weiterhin möchte ich Herrn Gerald Schröder für seine Unterstützung und Betreuung danken.

Inhaltsverzeichnis

1 Motivation und Zielsetzung **1**

1.1 Ziel der Arbeit 3

1.2 Aufbau der Arbeit 4

2 Anforderungen an unternehmensweite Client/Server-Anwendungen **7**

2.1 Grundlegende Begriffe 7

2.1.1 Ein Architekturmodell für Client/Server-Anwendungen 9

2.1.2 Alternative Client/Server-Anwendungsarchitekturen 11

2.2 Einsatz von Client/Server-Systeme in Unternehmen 13

2.3 Externe Qualitätsmerkmale unternehmensweiter Client/ Server-Anwendungen 13

3 Anforderungen an Client/Server-Entwicklungsumgebungen **17**

3.1 Entwicklung des Kriterienkatalogs 17

3.2 Anforderungen an die Systemarchitektur 18

3.2.1 Betriebssystemschnittstelle 19

3.2.2 Netzwerkschnittstelle 20

3.2.3 Datenbanksystemschnittstelle 20

3.2.4 Schnittstellen zu externen Anwendungen 21

3.2.5 Schnittstellen zu anderen Sprachen 22

3.2.6 Fenstersystemschnittstelle 22

3.3 Anforderungen an das Programmiermodell 23

3.3.1 Datenstrukturierungskonzepte 23

3.3.2 Strukturierungstechniken für die Programmierung im Kleinen 27

3.3.3 Strukturierungskonzepte für die Programmierung im Großen 29

3.3.4 Verteilungsabstraktion 31

3.3.5 Persistenzabstraktion 35

3.3.6 Massendatenverarbeitung 36

3.3.7 Transaktionen 37

3.3.8 Programmierung grafischer Benutzerschnittstellen 38

3.4 Kriterien für die Werkzeugwahl 39

3.4.1 Werkzeuge für die Analyse der Geschäftsprozesse 39

3.4.2 Werkzeuge zur Implementierung 39

3.4.3 Software-Management-Werkzeuge 41

3.5 Korrelation zwischen externen und internen Qualitätsmerkmalen 41

4 NATURAL: Eine 4GL-Entwicklungsumgebung 43
4.1 Die NATURAL-Systemarchitektur . . . 43
4.1.1 Betriebssystemschnittstelle . . . 44
4.1.2 Netzwerkschnittstelle . . . 44
4.1.3 Datenbanksystemschnittstelle . . . 44
4.1.4 Schnittstellen zu externen Anwendungen . . . 45
4.1.5 Schnittstelle zu anderen Sprachen . . . 45
4.1.6 Fenstersystemschnittstelle . . . 46
4.2 Das NATURAL-Programmiermodell . . . 46
4.2.1 Datenstrukturierungskonzepte . . . 46
4.2.2 Strukturierungstechniken für die Programmierung im Kleinen . . . 48
4.2.3 Strukturierungskonzepte für die Programmierung im Großen . . . 49
4.2.4 Verteilungsabstraktion . . . 52
4.2.5 Massendatenverarbeitung . . . 54
4.2.6 Transaktionen . . . 56
4.2.7 Programmierung grafischer Benutzerschnittstellen . . . 57
4.3 Entwicklungswerkzeuge . . . 59
4.3.1 Werkzeuge für die Analyse der Geschäftsprozesse . . . 59
4.3.2 Werkzeuge zur Implementierung . . . 60
4.3.3 Software-Management-Werkzeuge . . . 61

5 System R/3: Eine Entwicklungsumgebung für betriebliche Informationssysteme 63
5.1 Die Architektur des Systems R/3 . . . 63
5.1.1 Betriebssystemschnittstelle . . . 65
5.1.2 Netzwerkschnittstelle . . . 65
5.1.3 Datenbanksystemschnittstelle . . . 65
5.1.4 Schnittstellen zu externen Anwendungen . . . 66
5.1.5 Schnittstellen zu anderen Sprachen . . . 66
5.1.6 Fenstersystemschnittstelle . . . 66
5.2 Das ABAP/4-Programmiermodell . . . 67
5.2.1 Datenstrukturierungskonzepte . . . 67
5.2.2 Strukturierungstechniken für die Programmierung im Kleinen . . . 69
5.2.3 Strukturierungstechniken für die Programmierung im Großen . . . 71
5.2.4 Verteilungsabstraktion . . . 72
5.2.5 Persistenzabstraktion . . . 73
5.2.6 Massendatenverarbeitung . . . 73
5.2.7 Transaktionen . . . 76
5.2.8 Programmierung grafischer Benutzerschnittstellen . . . 78
5.3 Entwicklungswerkzeuge . . . 79
5.3.1 Werkzeuge zur Implementierung . . . 79
5.3.2 Software-Management-Werkzeuge . . . 81

6 Tycoon: Eine Entwicklungsumgebung für persistente Objektsysteme 83
6.1 Die Architektur des Tycoon-Systems . . . 83
6.1.1 Betriebssystemschnittstelle . . . 84
6.1.2 Netzwerkschnittstelle . . . 84
6.1.3 Datenbanksystemschnittstelle . . . 85

6.1.4 Schnittstellen zu externen Anwendungen 85
6.1.5 Schnittstelle zu anderen Sprachen 85
6.1.6 Fenstersystemschnittstelle 87
6.2 Das Tycoon-Programmiermodell 87
6.2.1 Datenstrukturierungskonzepte 87
6.2.2 Strukturierungskonzepte für die Programmierung im Kleinen 89
6.2.3 Strukturierungskonzepte für die Programmierung im Großen 91
6.2.4 Verteilungsabstraktion 92
6.2.5 Persistenzabstraktion 93
6.2.6 Massendatenverarbeitung 93
6.2.7 Transaktionen 95
6.2.8 Programmierung grafischer Benutzerschnittstellen 95
6.3 Entwicklungswerkzeuge 97

7 Zusammenfassung und Ausblick **101**
7.1 Bewertungsmöglichkeiten 101
7.2 Nutzung der Ergebnisse dieser Arbeit 102
7.3 Ausblick 104

Abbildungsverzeichnis

1.1 Paradigmenwechsel in der EDV 2

2.1 Client/Server-Computing 8
2.2 Ein nicht striktes Schichtenmodell für die Zerlegung einer Client/Server-Anwendung 9
2.3 Alternative Client/Server-Konfigurationen 11

3.1 Die strategischen Schichten einer Softwarearchitektur 19
3.2 Eine heterogene Datenbankumgebung 21
3.3 Multi-Threading Architektur für Server 29
3.4 Verteilungsabstraktion 32
3.5 Eine verteilte Transaktion 37

4.1 Die offene Architektur von NATURAL 44
4.2 Netzwerk-Integration in NATURAL 45
4.3 Unterprogrammtechniken in NATURAL 50
4.4 Entfernter Datenbankzugriff in NATURAL 53
4.5 NATURAL-RPC 54
4.6 NATURAL-DDM-Konzept 55
4.7 Sprachelemente zu Datenmanipulation in NATURAL 55
4.8 Dialog-Objekte in NATURAL 58

5.1 Schichtenarchitektur des Systems R/3 64
5.2 Kommunikationsprotokollschichten im System R/3 65
5.3 Sprachelemente zur Datenmanipulation in ABAP/4 75
5.4 Transaktionen im System R/3 76

6.1 Die Schichtenarchitektur des Tycoon-Systems 84
6.2 Das offene persistente Objektsystem Tycoon 86
6.3 Massendatenverarbeitung in Tycoon 94
6.4 Entwicklung grafischer Benutzerschnittstelle in Tycoon 96
6.5 Aufruf von TL-Funktionen aus StarView-Präsentation 97

7.1 Produktevaluation 102
7.2 Produktevaluation anhand der mit Interoperabilität positiv korrelierten, internen Qualitätsmerkmale 103

Kapitel 1

Motivation und Zielsetzung

Client/Server-Architekturen werden allgemein als ein wichtiger Meilenstein in der Entwicklung integrierter betriebswirtschaftlicher Anwendungen angesehen. Heute wird die Client/Server-Architektur nicht mehr als ferne Vision, sondern als konkrete Leitlinie für den Einsatz der Informationstechnologie zur Unterstützung der Geschäftsprozesse in Unternehmen begriffen [Hansen et al. 93]. In vielen Unternehmen wird inzwischen die tragende Rolle erkannt, die Client/Server-Anwendungen im Rahmen der Optimierung von Geschäftsprozessen spielen [Niemann 95].

Die Motivation und Rechtfertigung für Client/Server-Systeme ist sehr stark mit Unternehmensanforderungen verbunden. Die Gründe für Client/Server-Architektur sind mehr in geänderten Unternehmensstrukturen zu finden als bei der Technologieentwicklung. Als Ergebnis der Evolution im Einsatz von Informationstechnologie zur Unterstützung der Kerngeschäftsprozesse hat in den letzten zehn Jahren die verteilte Verarbeitung zunehmend an Bedeutung gewonnen. Diese Entwicklung ist in Abbildung 1.1 in Anlehnung an Simon Forge [Forge 94] zusammengefaßt. Dabei werden drei Architekturen für betriebswirtschaftliche Anwendungen unterschieden, die jeweils auch stark durch die zu unterstützenden Unternehmensstrukturen geprägt sind.

Zu Beginn des Einsatzes von DV-Systemen im kommerziellen Bereich wurden entsprechend der organisatorischen Zentralisation der Unternehmen große Systeme mit einem starken Zentralisierungsgrad benutzt. Der Benutzerkreis des Systems war auf wenige Spezialisten eingeschränkt, die organisatorisch in zentralen DV-Abteilungen konzentriert waren.

Zur Überwindung der Schwächen von Großorganisationen wurde in den 80er Jahren eine Unternehmensstrategie entwickelt, die zu einer Konzentration auf das Kerngeschäft und zur Bildung kleiner, weitgehend autonom operierender, an ökonomischen Zielen ausgerichteter Organisationseinheiten führte [Knolmeyer 92]. Begleitet wurde diese Umsetzung von der Entwicklung der individuellen Datenverarbeitung [Seibt 92]. Technische Grundlagen bildeten Arbeitsplatzrechner (PCs), neue, arbeitsplatzbezogene Standardsoftware und grafische Oberflächen. Eine Integration von zentralen Systemen und der individuellen DV fand nur durch Nutzung von Terminalemulation durch PCs statt. Die dezentral in Form von PCs und Workstations vorhandene Rechnerleistung wird bei einer ausschließlich arbeitsplatzbezogenen Nutzung nur unzureichend ausgelastet.

In den 90er Jahren sind die Unternehmen geprägt von der Optimierung der Geschäftsprozesse durch Veränderungen in der Aufbau- und Ablauforganisation. Basierend auf ganzheitlichen Geschäftsprozessen werden Arbeitsorganisationen mit entsprechender Qualifikation und Motivation des Personals gebildet [Buck-Emden et al. 93]. Optimie-

	Unternehmensstruktur	Hauptaufgabe der IT	Arbeitsweise der IT	Benutzerkreis
70er Jahre	Zentralisiert hierarchisch, statisch	Back-Office Unterstuetzung (Lohn- & Finanz-buchhaltung, Kunden-Datenbank)	Serielle Hintergrund-verarbeitung, zentrale Datenhaltung und -zugriff	Eingeschraenkt: Systemadministrator Datentypisten
80er Jahre	Strategische Geschaeftseinheiten werden in Funktionen organisiert, verschiedene Strukturen werden bei Geschaeftsuebernahmen verschmolzen	Automatisierung der Bueroarbeit, Entscheidungs-unterst tzung	Komplexe interaktive Anwendungen, einzelstehende PC-Anwendungen	Ausgeweitet: Sekraeterin, Managerin, Programmiererin
90er Jahre	Uebergang zu flexiblen, flachen, sich selbst organisierenden Strukturen, basierend auf Gesch ftsprozessen	Unterstuetzung aller Geschaeftsprozesse, technischer und strategischer Entscheidungen, fundamentale Rolle digitalisierter Dokumente	Verteilte Verarbeitung Integration der PC-Anwendungen	Universell: Einfache Angestellte bis Manager, ueberall und jederzeit besonders mobil und ausserhalb der Firma

Abbildung 1.1: Paradigmenwechsel in der EDV

rende Geschäftsprozesse und flache Hierarchien erfordern eine qualitativ anspruchsvolle Unterstützung an jedem Arbeitsplatz, um mit steigender Verantwortung des Einzelnen die zur Aufgabenerfüllung notwendigen Ressourcen bereitzustellen [Niemann 95]. Die unternehmensweite Integration von Informationen und Diensten wird durch eine kooperative Datenverarbeitung zwischen Mainframe-Welt und PC-Welt in einer Client/Server-Architektur realisiert.

Client/Server-Architekturen versprechen folgende Vorteile im Vergleich zu monolithischen Großrechnerlösungen, die auch das anhaltend hohe Interesse an Client/Server-Systemen für moderne betriebswirtschaftliche Anwendungssoftware erklären:

- Benutzerfreundlichkeit: Client/Server-Technologie erlaubt die Synthese der Vorteile zentraler Systeme und der benutzerfreundlichen Oberflächen von Arbeistplatzrechnern. Die Integration von operativen Systemen und Büroinformationssystemen durch grafische Oberflächen am Arbeitsplatz läßt zusätzliche Synergieeffekte erwarten.

- Standardisierung: Durch die Aufteilung von Anwendungen in eine Client- und eine Serverkomponente werden Entwickler gezwungen, klare und zunehmend vorgegebene Schnittstellen innerhalb einer Anwendung einzuhalten.

- Integration von Anwendungssystemen: Die Standardisierung von Client/Server- Systemen bewirkt, daß die technischen Voraussetzungen für eine Integration der unterschiedlichen betrieblichen Anwendungssysteme geschaffen werden.

- Skalierbarkeit: Bei steigenden Leistungsanforderungen kann sich das Gesamtssystem durch die Hinzunahme neuer Verarbeitungsknoten an die nachgefragte Leistung anpassen.

- Problemadäquate Verteilung der Systemlast durch Spezialisierung von Hardware und Software: Durch die Spezialisierung von Hard- und Systemsoftware im Bereich der Server werden deutliche Performancevorteile gegenüber Universalrechnern als Server erreicht. Zum Beispiel im Bereich der Datenbank-Server können Server konzipiert werden, die für die besonderen Aufgaben der transaktionsorientierten Datenbankanwendungen (bzw. deren auf dem Server implementiertem Datenbank-Teil) optimiert sind.

- Verbesserte Kapazitätsauslastung teurer Ressourcen: Der Einsatz von Client/Server-Systemen erlaubt die Zentralisation von Ressourcen (z.B. Speicherperipherie, Kommunikationsendgeräte, Softwarewartung etc.).

- Datensicherheit: Durch den Einsatz von Client/Server-Systemen besteht die Möglichkeit, Systeme in eine vertrauenswürdige Serverkomponente und eine nicht vertrauenswürdige Clientkomponente aufzuspalten. Die Serverkomponente des Systems kann durch Sicherheitsmechanismen gegen unberechtigte Zugriffe geschützt werden. Die auf dem Client verbleibenden Teile des Systems können entsprechend der Risikostrategie des Betreibers behandelt werden.

Zusammenfassend ermöglichen Client/Server-Architekturen, eine neue Klasse von Anwendungen für das gesamte Unternehmen zu realisieren, die auch als unternehmensweite Client/Server-Anwendungen oder als Enterprise-Client/Server bezeichnet werden.

Andererseits erfordert der Übergang von zentralisierten Time-Sharing-Systemen zu unternehmensweiten Client/Server-Architekturen radikale Änderungen nicht nur der Systemarchitektur, sondern auch der Analyse-, Design- und Implementierungsmodelle und der sie unterstützenden Werkzeuge, die erheblich von den traditionellen Terminal- und Mainframe-Werkzeugen abweichen. Als Beispiele seien hier genannt: visuelle Werkzeuge zur Entwicklung grafischer Oberflächen, schnelle Anwendungsentwicklung (rapid prototyping), wiederverwendbare, mit objektorientierten Techniken modellierte Komponenten, Mechanismen zur Client/Server-Interprozeßkommunikation, Firewalls in offenen Netzen, Produkte zum Internetworking.

Daher besteht bei Organisationen ein starkes Interesse an integrierten Entwicklungsumgebungen als Plattformen für die effiziente Realisierung unternehmensweiter Client/ Server-Anwendungen. Diese Werkzeuge sind Gegenstand der vorliegenden Arbeit.

1.1 Ziel der Arbeit

Dieser Arbeit sollte Leitfaden für die Evaluation von Client/Server- Entwicklungsumgebungen anbieten. Dabei bildet die Erstellung eines technischen Kriterienkatalogs das Hauptziel der Arbeit. In diesem Katalog sollten technischen Kriterien basierend auf Informatik-Fachliteratur erläutert und ihre Relevanz für die Entwicklung unternehmensweiter Client/Server-Anwendungen diskutiert werden. Der entstehende Katalog sollte als eine Schritt-für-Schritt Anleitung bei der Bewertung von Client/Server-Entwicklungsumgebungen eingesetzt werden können.

In dieser Arbeit wird dieser Katalog zu Gegenüberstellung der Systeme NATURAL (Software AG, Darmstadt), SAP R/3 (SAP AG, Walldorf) und des prototypischen Tycoon-Systems (Universität Hamburg) als Plattformen für die effiziente Realisierung unternehmensweiter Client/Server-Anwendungen verwendet. Alle drei Systeme versprechen eine

nahtlose Integration von Anwendungen, Daten, Benutzerschnittstellen und weiteren Systemdiensten in (verteilten) Systemen praktisch aller Größenordnungen.

NATURAL ist die Entwicklungsumgebung der Software AG, einem der größten deutschen Software-Hersteller. Diese Entwicklungsumgebung ist mit der 4GL Sprache NATURAL und dem Datenbanksystem ADABAS auf dem Mainframe-Markt erfolgreich geworden. Seit dem ersten Release im Jahre 1979 setzen ca. 4000 Kunden weltweit NATURAL als ihre Entwicklungsumgebung ein. Dadurch kann NATURAL als ein De-facto-Standard für Systeme der 4. Generation betrachtet werden. NATURAL steht inzwischen auf verschiedenen Plattformen und Datenbanken zur Verfügung. Unterstützt durch PC-basierte Programmentwicklung, Middlewaretechnologie auf der Ebene von Datenbanken und durch Netzwerkkommunikation, die für die Client/Server-Integration in heterogenen Netzen nötig ist, bildet NATURAL auch eine Plattform zur Entwicklung von Client/Server-Anwendungen.

Das System R/3 ist die Client/Server-Software der SAP AG, die im Jahr 1992 zur Unterstützung der Rightsizing-Anforderungen ihrer Kunden als Nachfolger der im Markt verbreiteten SAP-Mainframe-Software R/2 entwickelt wurde. Diese Software ist nach dem Client/Server-Ansatz konzipiert und nicht mehr auf zentralen Großrechnern lauffähig. Die Bedeutung der SAP-Software als ein De-facto-Standard im Bereich betriebswirtschaftlicher Standardsoftware läßt einen erheblichen Einfluß dieser Entscheidung auf die Gestaltung zukünftiger DV-Infrastrukturen erwarten.

Das Tycoon-System ist eine integrierte, persistente und polymorphe Programmierumgebung, die im Rahmen eines europäischen ESPRIT Forschungsprojektes am Arbeitsbereich Datenbanken und Informationssysteme der Universität Hamburg zwischen 1992 und 1995 entwickelt und implementiert wurde. Durch leistungsfähige Kommunikationsmechanismen und äußerst flexible Möglichkeiten zur Partitionierung von Tycoon-Anwendungen in heterogenen Netzen kann Tycoon auch als Plattform für die effiziente Realisierung unternehmensweiter Client/Server-Anwendungen eingesetzt werden.

1.2 Aufbau der Arbeit

In einem ersten Schritt werden die grundlegenden Begriffe der Client/Server-Technologie vorgestellt. Basierend auf diesen Grundlagen wird der Einsatz von Client/Server-Technologie in Unternehmen untersucht und ein Anforderungskatalog (externe Qualitätsmerkmale) an Client/Server-Anwendungen für die Unternehmen erstellt (Kapitel 2).

In einem zweiten Schritt (Kapitel 3) werden ausgehend von diesem Anforderungskatalog für Client/Server-Anwendungen konkrete technische Anforderungen an Entwicklungsumgebungen für Client/Server-Anwendungen abgeleitet. Diese Anforderungen (interne Qualitätsmerkmale) betreffen sowohl die Architektur, das Programmiermodell als auch die Werkzeuge der Entwicklungsumgebung. Insgesamt werden ca. 100 interne Qualitätsmerkmale identifiziert, definiert und als konkrete Checkliste formuliert. Unabhängig von den konkret untersuchten Systemen wird am Ende von Kapitel 3 in einer Korrelationstabelle die Beziehung zwischen internen und externen Qualitätsmerkmalen dargestellt.

Die dann folgenden drei Kapitel (Kapitel 4-6) nutzen diese Checkliste zur detaillierten, nachvollziehbaren Analyse der Systeme NATURAL, SAP R/3 und Tycoon als Plattformen für die effiziente Realisierung unternehmensweiter Client/Server-Anwendungen. Die Einzelergebnisse der drei Kapitel werden außerdem übersichtlich in Anhang xx in tabellarischer Form gegenübergestellt.

Die Arbeit endet in Kapitel 7 mit einer prägnanten textuellen Zusammenfassung der Detailergebnisse und eine Diskussion der Bewertungsmöglichkeiten von Client/Server-Entwicklungsumgebungen mit Hilfe der in dieser Arbeit entwickelten Methodik.

Kapitel 2

Anforderungen an unternehmensweite Client/Server-Anwendungen

Eine einheitliche Terminologie für Client/Server-Konzepte hat sich noch nicht allgemein durchgesetzt. Viele Begriffe, wie Client/Server-Modell, Client/Server-Umgebung usw., werden synonym benutzt. Die Umsetzung der Client/Server-Idee in konkrete Systeme wird als Client/Server-Technik oder Client/Server-Technologie bezeichnet. Werden Anwendungssysteme nach diesem Gedanken implementiert, werden sie als Client/Server-Systeme, Client-/Server-Anwendungen oder Client/Server-Software bezeichnet. Die Art der Datenverarbeitung mittels dieser Systeme wird dann als Client/Server-Computing bezeichnet.

Die Beschreibung der Charakteristika unternehmensweiter Client/Server-Anwendungen und die Aufstellung eines Anforderungskatalogs setz die klare Definition von Client/Server-Anwendungen und der relevanten Grundbegriffe voraus. Deswegen befaßt sich der erste Teil dieses Kapitels mit der Definition von Client/Server-Anwendungen und stellt alternative Architekturen für Client/Server-Anwendungen vor. Basierend auf diesen Grundlagen werden im zweiten Teil des Kapitels die externen Qualitätsmerkmale von Client/Server-Lösungen für unternehmensweite Anwendungen beschrieben.

2.1 Grundlegende Begriffe

Der Begriff Client/Server beschreibt eine Anwendungsarchitektur mit einer Rollenaufteilung in Dienstfordernde (Clients) und Diensterbringende (Server), wobei dieses Modell voll rekursiv ist. Diese Anwendungsteile können dabei auf unterschiedliche Systemplattformen verteilt werden. Grundgedanke des Client/Server-Prinzips ist dabei die Schaffung wiederverwendbarer Dienste, die dezidierte Leistungen erbringen und spezielle Leistungsfähigkeiten von Systemplattformen optimal ausnutzen [Schmitt 93]. Ausgehend von diesem Grundgedanken zeichnen sich die auf einer Client/Server-Architektur basierenden Systeme durch folgende Merkmale aus:

- Ein System wird in mindestens zwei Subsysteme aufgeteilt. Die Subsysteme sind auf zwei oder mehrere Plattformen verteilt.
- Die als Server bezeichneten Subsysteme erbringen Dienstleistungen für andere Subsysteme.

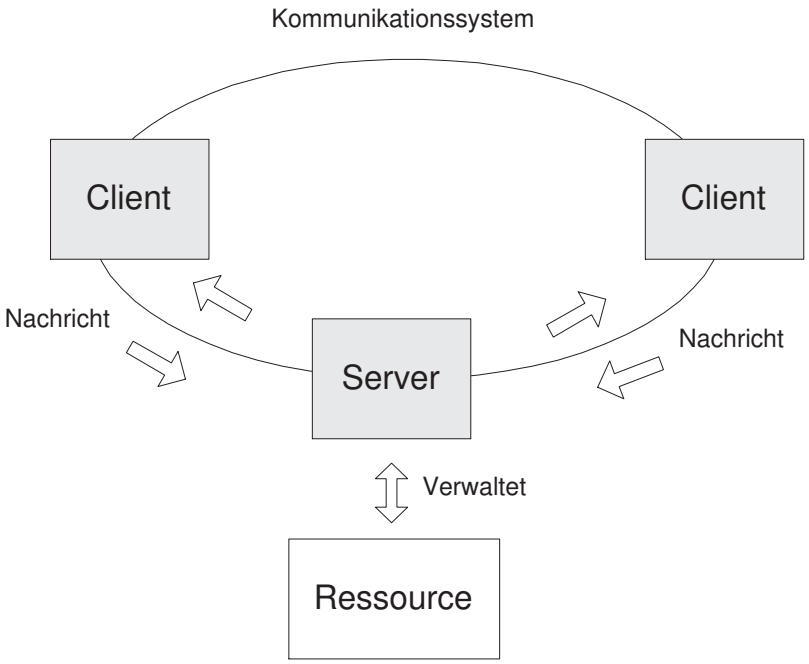

Abbildung 2.1: Client/Server-Computing

- Die Server verwalten eine gemeinsam genutzte Ressource, die ein im Sinne der Objektorientierung gekapseltes Objekt (Daten und Zugriffsmethoden) darstellt.

- Die Clients nutzen die Dienstleistungen von Servern, indem sie Dienste von den Servern aktiv anfordern.

- Clients und Server kommunizieren über Netzwerke und Middleware[1]. Die Hardware, die dem Client die Zusammenarbeit mit dem Server ermöglicht, kann ein LAN (*Local Area Network*) oder/und ein WAN (*Wide Area Network*) sein.

- Client und Server sind Rollen. Server können selbst wieder Dienste anderer Server benutzen und treten dann in der Rolle eines Clients auf.

- Eine Anwendung ist Client/Server und nicht die zugrundeliegende Hardwarekonfiguration.

Abbildung 2.1 stellt diese Merkmale grafisch dar.

Zusammengefaßt bestehen Client/Server-Anwendungen aus Komponenten, die als Dienstnutzer auf Client-Rechnern installiert sind, und Anwendungsdiensten, die auf Server-Rechnern implementiert werden und von mehreren Anwendungen genutzt werden können. Die zur Unterstützung eines Geschäftsprozesses erforderlichen Funktionen und Daten werden also in Form von Diensten (Server) abgebildet. Diese Dienste können nun von den Dienstnutzern (Clients) verwendet werden.

[1]Software, die eine Verbindung über ein Netz vermittelt, wird Middleware genannt.

Abbildung 2.2: Ein nicht striktes Schichtenmodell für die Zerlegung einer Client/Server-Anwendung

2.1.1 Ein Architekturmodell für Client/Server-Anwendungen

Ein wichtiger Punkt der Client/Server-Architektur ist die Art und Weise der Zerlegung einer Anwendung in Teilkomponenten und die Verteilung dieser Teile auf verschiedene Plattformen. Eine optimale Verteilung einer Anwendung gestattet es, Teile der Anwendung auf den zur Verfügung stehenden Rechnerklassen so abzubilden, daß jede Rechnerklasse ihrer spezifischen Bauweise und Funktionalität entsprechend eingesetzt wird. Der Inhalt der Teilkomponente hängt normalerweise von den Anforderungen und spezifischen Eigenschaften der Anwendung ab. Grundsätzlich können fünf Komponenten einer datenintensiven Anwendung unterschieden werden, die nachfolgend näher erläutert werden:

- Präsentation
- Steuerung
- Anwendungslogik
- Datenverwaltung
- Datenhaltung

Wie in Abbildung 2.2 dargestellt ist, bilden diese Komponenten ein nicht striktes Schichtenmodell.

Präsentation

Die Präsentationskomponente besteht aus Funktionen zum Anzeigen fachlicher und technischer Daten am Bildschirm bzw. zu deren Ausgabe und Druck. Die Art der Bildschirmgestaltung wird als grafische Benutzerschnittstelle (Graphical User Interface, GUI) bezeichnet. Die wesentlichen Eigenschaften grafischer Benutzerschnittstellen sind:

- Anwendung des Prinzips der direkten Manipulation
- Prinzip des What-you-see-is-what-you-get (WYSIWYG), d.h. das Layout von Dokumenten unterscheidet sich am Bildschirm nicht vom Ausdruck

- Einsatz der Fenstertechnik, d.h. die Aufteilung des physischen Bildschirmes in mehrere logische Bildschirme
- Standardisierte Kommunikationsmechanismen zum benutzergesteuerten Datenaustausch (*drag and drop*) zwischen Anwendungen

Steuerung

Die Steuerungskomponente beinhaltet die Ablauflogik der Geschäftsprozesse. Sie steuert den Dialog mit dem Benutzer und reagiert auf seine Eingaben. Die Steuerungskomponente benutzt die Funktionen der Präsentationsschicht, um Ausgaben auf dem Bildschirm oder dem Drucker durchzuführen. In datenbankorientierten Systemen besteht die Hauptaufgabe der Dialogsteuerung in der Kontrolle der Dialogschritte, die zur Abarbeitung von Transaktionen notwendig sind, und in der Gewährleistung von integritätssichernden Maßnahmen [Schmitt 93]. Benutzerfreundliche Anwendungssysteme erfordern, daß Integritätsverletzungen nicht erst beim Einbringen in die Datenbank aufgedeckt und verhindert werden, sondern daß bereits in der Benutzerschnittstelle Mechanismen vorgesehen sind, die Integritätsverletzungen vermeiden oder im Fehlerfall Hilfestellung bei der Lösung des Integritätskonflikts geben. Typische Oberflächenelemente sind: Auswahllisten (*Combo-Box*) bei referenzieller Integrität, 1-aus-N-Auswahlknöpfe (*Radio-Buttons*) bei Aufzählungstypen, kurze Hilfetexte zu den Domänen, über denen Attribute definiert sind.

Anwendungslogik

Die Anwendungslogik realisiert die spezifische Funktionalität der Anwendung [Schmitt 93]. In der Anwendungslogik sind die fachlichen Funktionen des Anwendungssystems abgebildet. Beispiele für fachliche Funktionen sind Umsätze berechnen oder Angebot kalkulieren. Die Steuerungsschicht benutzt die fachlichen Funktionen, um die mit Hilfe der Präsentationsschicht darzustellenden fachlichen Daten aufzubereiten.

Datenverwaltung

Die Datenverwaltungsschicht besteht aus Integritätsregeln, die an die Datenstrukturen gebunden sind und die bei der Manipulation von Daten immer zu berücksichtigen sind. Beispiele für diese Regeln sind die referenzielle Integrität, Domänen und anwendungsspezifische Integritätsbedingungen. Die Steuerungsschicht benutzt die Datenverwaltungsschicht, um Eingaben des Benutzers zu prüfen. Die Schicht der Anwendungslogik benutzt die Datenverwaltung, um die Konsistenz der Daten bei der Speicherung zu gewährleisten.

Datenhaltung

Die Datenhaltungskomponente einer Anwendung verwaltet die für ein Anwendungssystem benötigten und durch mehrere Prozesse benutzten, persistenten Daten und realisiert den Zugriff auf diese Daten. Die Datenhaltungsschicht wird in der Regel durch ein Datenbankmanagementsystem realisiert. Die Funktionen der Datenhaltung entsprechen somit denen des Datenbanksystems. Moderne Datenbanksysteme bieten jedoch auch weitergehende Funktionen zur Datenverwaltung und Realisierung der Anwendungslogik. Eine Differenzierung in der Client/Server-Architektur ist trotzdem sinnvoll, weil diese Funktionen mit datenbanksystemspezifischen Mechanismen realisiert sind, die nicht immer mit der anwendungsspezifischen Funktionalität der Anwendungslogik übereinstimmen.

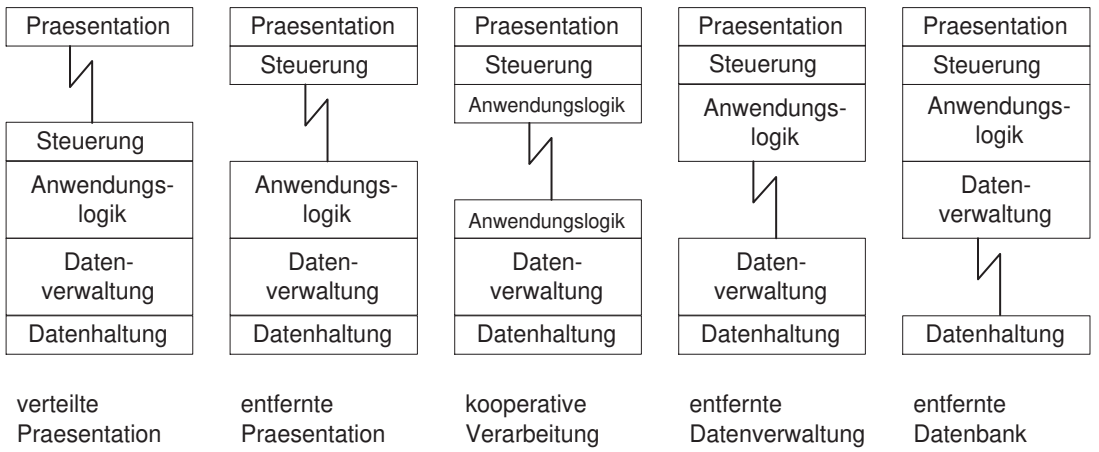

Abbildung 2.3: Alternative Client/Server-Konfigurationen

Welche Teile einer Client/Server-Anwendung auf dem Client und welche auf dem Server installiert werden sollten, hängt von den speziellen Anforderungen der Anwendung ab.

2.1.2 Alternative Client/Server-Anwendungsarchitekturen

Anhand des oben dargestellten Schichtenmodells von Anwendungen können die verschiedenen Möglichkeiten der Verteilung von Anwendungen nach dem Client/Server-Prinzip dargestellt werden. Die zentrale Idee der Client/Server-Architektur besteht darin, eine Anwendung an einer Stelle, z.B. an den oben beschriebenen Schnittstellen zwischen den einzelnen Schichten, aufzuteilen. Die in einen Client- und einen Server-Teil zerlegte Anwendung wird auf Verarbeitungsknoten in einem Netzwerk verteilt und die beiden Komponenten werden durch ein Kommunikationssystem verbunden. Andererseits sind Client/Server-Architekturen sehr eng mit der Netzwerkproblematik verbunden. Die Übertragungsleistung und Flexibilität eines Netzwerkes hängt generell von mehreren Faktoren ab. Das sind vor allem die Anzahl der Netzteilnehmer, deren räumliche Verteilung, die Kommunikationsfrequenz und natürlich auch die Art der Anwendung. Nach einem Vorschlag von Niemann [Niemann 95] lassen sich die in Abbildung 2.3 dargestellten Typen von Client/Server-Konfigurationen unterscheiden.

Verteilte Präsentation

Die verteilte Präsentation zeichnet sich durch eine Realisierung der Präsentationsfunktionen auf dem Client aus. Alle anderen Komponenten werden auf dem Server realisiert. Die Steuerungskomponente verwendet eine Netzwerkschnittstelle, um die Funktionen der Präsentationsschicht aufzurufen. Die Prüfung der Benutzereingaben erfolgt durch Funktionen der Datenverwaltung auf dem Server, die von der Steuerungsschicht aufgerufen werden. Die verteilte Präsentation kann bei reinen Auskunfts- und Anzeigesystemen sinnvoll sein.

Entfernte Präsentation

Neben grafischen Präsentationsfunktionen werden auch Dialogsteuerungslogik und Prüffunktionen auf dem Client installiert. Die Datenverwaltung kann sowohl auf dem Client als auch auf dem Server realisiert werden. Falls sie nur auf dem Server installiert wird, werden Zugriffe über das Netz auf die Integritätsregeln zwecks Prüfung der Benutzereingaben notwendig. In diesem Fall liegt die Steuerung im Rahmen einer ereignisgesteuerten Oberfläche auf dem Client.

Kooperative Verarbeitung

Bei der kooperativen Verarbeitung wird die Anwendungslogik zwischen Client und Server geteilt. Teile der Anwendungslogik werden auf dem Client installiert, während andere Teile als Anwendungsserver auf dem Server realisiert werden. Diese Art der Verteilung findet sich bei großen Client/Server-Anwendungen mit vielen Benutzern und weiträumiger Verteilung. Die Anwendungskomponente kann an einer Stelle aufgeteilt werden, an der wenig Datenvolumen zu bewegen ist.

Entfernte Datenverwaltung

In diesem Verteilungsmodell werden Präsentation, Steuerung und Anwendungslogik auf dem Client-Rechner installiert, während die Datenverwaltung und die Datenhaltung auf dem Server-Rechner liegen. Die Speicherung der an die Datenstruktur gebundenen Geschäftsregeln an einer zentralen Stelle sorgt dafür, daß alle Anwendungssysteme die Geschäftsregeln in gleicher Form benutzen.

Entfernte Datenbank

Die Datenbank wird auf dem Server-Rechner installiert. Die Datenverwaltungsschicht kommuniziert mit der Datenbank über eine Programmierschnittstelle, die auf dem Client-Rechner zu Verfügung steht. Der Zugriff auf die entfernte Datenbank erfolgt über die Verteilungsschnittstelle(*Remote Database Access*).

Verteilte Datenhaltung

Das Verteilungsmodell der verteilten Datenbank entspricht weitgehend dem des Zugriffs auf eine entfernte Datenbank. Im Unterschied hierzu ist auf jedem Rechner ein Datenbankmanagementsystem installiert. Die Verteilung der Daten wird vollständig vom Datenbanksystem übernommen.

Mehrstufige Client/Server-Architektur

Bei den bisher vorgestellten Client/Server-Architekturen wurde die Anwendung an einer Stelle aufgeteilt. Unterteilt man die Anwendung an mehreren Stellen, so entsteht eine mehrstufige Client/Server-Architektur. Dabei können die unterschiedlichsten Anwendungs- und Systemdienste auf die für ihre Aufgabe jeweils am besten geeigneten Server verlagert werden. So können z.B. verarbeitungsintensive Teile der Anwendungslogik auf eigenständige Verarbeitungsknoten ausgelagert werden. Es besteht auch die Möglichkeit der redundanten Implementierung einzelner Komponenten auf eigenen Knoten zur Erhöhung der

Ausfallsicherheit. Bei mehrstufigen Client/Server-Architekturen wird besonders der Vorteil der Skalierbarkeit von Client/Server-Systemen deutlich. Steigende Leistungsanforderungen an das Gesamtsystem können durch das Einfügen neuer Verarbeitungsknoten in den Verbund erfüllt werden.

Die oben vorgestellten Verteilungs- und die daraus abgeleiteten Architekturformen sind nur ein Musterbeispiel für eine mögliche Erstellung von Client/Server-Anwendungen. Es sind natürliche weitere Verteilungsformen denkbar.

2.2 Einsatz von Client/Server-Systeme in Unternehmen

Die schnelle, vollständige und konsistente Verfügbarkeit betriebswirtschaftlich relevanter Informationen ist heute das Ziel jedes Unternehmens. Daher befinden sich viele Unternehmen in einer Veränderungsphase der Aufbau- und Ablauforganisation. Zur Optimierung der Geschäftsprozesse werden Organisationen mit flachen Hierarchien und steigender Verantwortung der einzelnen Mitarbeiter eingerichtet. Neben dieser organisatorischen Veränderung werden neue Anforderungen an die Informations- und Kommunikationstechnologie gestellt. Die Aufgaben der betriebswirtschaftlichen Anwendungssysteme beschränken sich nicht mehr auf die Automatisierung isolierter Vorgänge in Logistik und Finanzwirtschaft [Buck-Emden et al. 93]. Vielmehr besteht die Aufgabe der Informationstechnologie darin, fortschrittliche Lösungen bereitzustellten, die die neu gestalteten Geschäftsprozesse mit verbesserter Qualität unterstützen können. Es werden Lösungen in Form von Desktop-Anwendungen, Fremdsystemen und eigenentwickelten Anwendungen erforderlich, die auf die Geschäftsprozesse ausgerichtet sind und die das gesamte Spektrum betriebswirtschaftlicher Informationsflüsse abdecken.

Durch Integration von Desktop-Anwendungen, die auf dem isolierten PC zur Verfügung stehen, mit den individuell entwickelten Systemen kann eine durchgängige Verarbeitung der Geschäftsprozesse garantiert werden. Damit können Desktop-Anwendungen zur Abdeckung eines Teil der Funktionalität der Geschäftsprozesse genutzt werden. Die unternehmensweite Lösung sollte auch eine Infrastruktur für den internen, externen und firmenübergreifenden Informationsaustausch anbieten. Zusammengefaßt stellen die Unternehmen folgende Anforderungen an ihre unterstützende Informationstechnologie:

- Integration aller Ressourcen, Hardware und Software
- optimierte Ressourcennutzung, z.B. PCs und Standardsoftware
- Vermeiden der redundanten Abbildung von Funktionalitäten
- schnelle Anpassung an organisatorische Änderungen und steigende Anforderungen
- schneller und einfacher Zugang zu Daten
- leichte Mensch-Maschine-Interaktio.

2.3 Externe Qualitätsmerkmale unternehmensweiter Client/ Server-Anwendungen

Durch den Einsatz der im ersten Teil dieses Kapitel vorgestellten Client/Server-Lösungen kann eine unternehmensweite Integration von Diensten erreicht werden. Unternehmensweites Client/Server-Computing führt Organisationen über Abteilungs-Client/Server und

Gateway-Produkte hin zur totalen Integration der abteilungsweiten in unternehmensweite Informationssysteme und ermöglicht eine neue Klasse von Anwendungen für das gesamte Unternehmen [Rosenhagen 95]. Alle Rechner des Unternehmens werden physisch an die unternehmensweite Netzwerkinfrastruktur angeschlossen. Die Arbeitsplätze werden mit Arbeitsplatzrechnern ausgestattet, d.h. Terminals werden vollständig durch PCs ersetzt. Anwendungen werden nach dem Client/Server-Prinzip entwickelt, Hosts treten in die Rolle unternehmensweiter Server. Nicht nur die physische Integration aller Rechner der Unternehmung im Sinne eines Anschlusses an ein LAN oder WAN, sondern auch die Verwendung offener Protokolle auf allen Ebenen kennzeichnet dieses Szenario. Unabhängig von der Rechnerhardware oder den Betriebssystemen werden einheitliche Protokolle zur Kommunikation verwendet. Neben den transportorientierten Protokollen sind auch die anwendungsorientierten Protokolle, z.B. Dateidienste, Datenbankzugriff, E-Mail, grafische Präsentation, unternehmensweit vereinheitlicht. Alle von Servern angebotenen Dienste werden systemweit transparent den Clients angeboten. Darüber hinaus werden diese Protokolle auch über die Unternehmensgrenzen hinaus von der Mehrzahl aller Rerchneranwender benutzt, so daß ein unternehmensübergreifender Austausch möglich ist. Externe Kommunikationsbeziehungen des Unternehmens werden durch die offene Infrastruktur unterstützt. Am Arbeitsplatz werden alle Informationsfunktionen durch ein einziges multimediales Endgerät mit einer einheitlichen Bedieneroberfläche unterstützt. Ausgehend von dieser allgemeinen Beschreibung unternehmensweiter Client/Server-Anwendungen können die folgenden externen Qualitätsmerkmale dieser Anwendungen hervorgehoben werden:

Portabilität

Portabilität ist die Möglichkeit der einfachen Übertragung von Anwendungen auf eine andere systemtechnische Basis (Hardware, Betriebssoftware, Datenbank, Netzwerk usw.) [Heinrich, Roithmay 89]. Die Verteilung der Anwendungsdienste auf der heterogenen Hardware- bzw. Softwarelandschafeten unternehmensweiter Client/Server-Anwendungen setzt ihre Portabilität voraus. Eine einmal realisierte Anwendung muß gegebenenfalls auf den Plattformen unterschiedlicher Hardwarehersteller lauffähig sein. Dies gilt besonders für die Präsentationskomponente der Anwendung. Die grafische Oberfläche muß auf den verschiedenen Frontend- Plattformen, wie z.B. Windows, Präsentation Manager oder Motif, einheitlich darstellbar sein. Darüber hinaus realisieren die Anwendungsdienste betriebswirtschaftliche Abläufe, deren Nutzungsdauer viel länger ist als die kurzen Innovationszyklen von Hardwarekomponenten, Betriebssystemen und Datenbanken. Diese Eigenschaften erfordern die Plattformunabhängigkeit dieser Anwendungen.

Interoperabilität

Interoperabilität im Sinne der Möglichkeit der Kommunikation zwischen heterogenen Systemen ist die Voraussetzung für die Realisierung der Kooperation der verteilten Dienste und der Integration aller Anwendungen in Client/Server-Architekturen. Clients, Anwendungs-, System- und Hardwaredienste müssen über mehrere heterogene Rechnerplattformen hinweg kooperieren. Diese Komponenten müssen durch Schnittstellen- und Realisierungsstandards aufeinander abgestimmt werden, um die erforderliche Interoperabilität zu gewährleisten. Ohne Interoperabilität von Anwendungen ist eine Integration der Anwendungssysteme nicht denkbar. Dies beinhaltet viel mehr als physische Kommunikation auf der Netzebene. Um eine vollständige Integration auf der Ebene von Anwendungssystemen zu erreichen, ist es zum Beispiel erforderlich, daß Ergebnisse, die von einem Dienst geliefert

werden, von den Dienstnutzern weiterverarbeitet werden können. Von den Unterschieden in bezug auf Codierungen, Formate oder Datentypen sollte durch die Schnittstellen abstrahiert werden können. Sicherheit und Konsistenz müssen beim Zugriff auf Dienste gewährleistet sein. Schließlich sollte die Client/Server-Anwendung mit Hilfe der Schnittstellen unabhängig von der Art der physischen Verteilung gehalten werden.

Kombinierbare Anwendungsbausteine

Wirklich verteilte Anwendungen entstehen, wenn neben der Datenbank auch Funktionen der Anwendung auf dem Server implementiert werden. Hierzu muß die Anwendung in Bausteine zerlegt werden, die untereinander über wohldefinierte Schnittstellen gekoppelt sind. Wenn es bei der Spezifikation dieser Bausteine gelingt, fachliche Zusammenhänge aus den Geschäftsprozessen des Unternehmens in Form von Datenstruktur und darauf operierenden Funktionen zu kapseln, so können die entstehenden Bausteine in mehr als einer Teilanwendung verwendet werden. Damit besteht die Anwendung aus Bausteinen, die Teilschritte der Geschäftsprozesse realisieren. Die Auslagerung mehrfach verwendbarer Funktionen und Daten beseitigt Redundanz und schafft Schnittstellen zur Verteilung.

Skalierbarkeit

Unternehmensweite Anwendungen müssen sich den wachsenden Anforderungen der Organisation flexibel anpassen können. Bei veränderten Lastprofilen, z.B. bei steigender Benutzerzahl oder beim Einsatz zusätzlicher Anwendungen, sollte die Anwendung horizontal und vertikal erweiterbar sein. Horizontale Erweiterung bedeutet das Hinzufügen zusätzlicher Client-Arbeitsplätze mit quantitativer Performanzsteigerung. Vertikale Erweiterung, d.h. die Migration zu leistungsfähigen Servern oder Multi-Servern, ermöglicht eine qualitative und quantitative Erweiterung des Systems in Hinblick auf den funktionalen Leistungsumfang.

Unterstützung grafischer Benutzeroberflächen

Ein zentrales Merkmal unternehmensweiter Client/Server-Anwendungen ist der Einsatz von grafischen Oberflächen mit Menü- und Fenstertechniken. Grafische Benutzerschnittstellen unterstützen den Zugang zu den unternehmensweiten Diensten und Informationsbeständen, integrieren Standardsoftware und ermöglichen die Unterstützung der Arbeitsabläufe in flachen Hierarchien durch Techniken des Workflow-Managements. Der Einsatz von grafischen Oberflächen erfordert eine ausreichende Unterstützung der Anwendung.

Kapitel 3

Anforderungen an Client/Server-Entwicklungsumgebungen

Die in Kapitel 2.4 genannten Anforderungen an unternehmensweite Client/Server- Anwendungen, wie z.B. Portabilität, Interoperabilität und Skalierbarkeit, ziehen konkrete Konsequenzen für die bei ihrer Softwarerealisierung eingesetzten Plattformen nach sich. Diese Konsequenzen werden in diesem Kapitel untersucht und die daraus folgenden Anforderungen beschrieben. Diese Beschreibung liefert einen Kriterienkatalog für die systematische Gegenüberstellung der Plattformen NATURAL, System R/3 und Tycoon.

Die Beschreibung der Untersuchungsmethode zur Aufstellung des Kriterienkatalogs bildet den einleitenden Abschnitt dieses Kapitels.

3.1 Entwicklung des Kriterienkatalogs

Ein systematischer Vergleich der Entwicklungsumgebungen setzt strukturierte und klar definierte Kriterien voraus. Eine naheliegende Methode zur Erstellung des Kriterienkatalogs wäre die Auswahl der in Abschnitt 2.3 erfaßten Anforderungen an unternehmensweite Client/Server-Anwendungen als Evaluationskriterien. Diese Anforderungen, die hier als **externe Qualitätsmerkmale** bezeichnet werden, eignen sich jedoch nicht als Evaluationskriterien, weil die externen Merkmale

- schwer zu quantifizieren,
- schwer weiter zu zergliedern und
- schwer gegeneinander abzugrenzen sind.

Eine alternative Untersuchungsmethode orientiert sich an den zugrundeliegenden Konzepten der Entwicklungsumgebungen, die hier als **interne Qualitätsmerkmale** bezeichnet werden. Diese internen Qualitätsmerkmale, wie z.B. die Systemarchitektur, können aus Konzepten der Programmiersprachen und der Softwaretechnik abgeleitet werden. Diese Vorgehensweise zur Bewertung komplexer Softwareentwicklungsplattformen findet sich in ähnlicher Form auch in der Literatur (z. B. [Scotts 82] und [Mühlhäuser 86]).

Der in dieser Arbeit entwickelte Kriterienkatalog interner Qualitätsmerkmale führt aufgrund der hohen Komplexität der zu vergleichenden Systeme zu einer sehr großen Zahl an Merkmalen. Durch Gruppieren der Merkmale in Klassen und durch eine hierarchische Gliederung ist es in dieser Arbeit gelungen, trotzdem einen überschaubaren Katalog zu

erstellen. Ein wichtiger Beitrag dieses Katalogs ist die Diskussion der Korrelation zwischen internen und externen Qualitätsmerkmalen. Dabei wird jeweils ausgehend von der Definition eines internen Qualitätsmerkmals seine Relevanz für die Unterstützung externer Qualitätsmerkmale analysiert. Im abschließenden Kapitel der Arbeit ist dann ausgehend von diesen Korrelationen eine systematische Gegenüberstellung der Entwicklungsumgebungen in Hinblick auf die besonderen Anforderungen unternehmensweiter Client/Server-Anwendungen möglich.

Diese Vorgehensweise kann wie folgt zusammengefaßt werden:

- Identifikation und Definition interner Qualitätsmerkmale der Systeme in Anlehnung an die Softwareliteratur (z.B. polymorphe Typisierung)
- Beschreibung der drei Systeme durch Ausprägungen dieser Qualitätsmerkmale (s. Kapitel 4, 5 und 6)
- Falls erforderlich, Präzisierung oder weitere Verfeinerung der Qualitätsmerkmale (z.B. das Einführen von Subtypisierung als Verfeinerung des internen Qualitätsmerkmals polymorphe Typisierung)
- Strukturierung der zusammengehörenden internen Qualitätsmerkmale zu Gruppen und ihre hierarchische Gliederung
- Feststellung und Formulierung der Korrelation zwischen internen und externen Qualitätsmerkmalen

Der auf diese Art und Weise entstandene Kriterienkatalog ist zum großen Teil aus der Literatur extrahiert, daher kann er nicht nur für die drei in dieser Arbeit untersuchten Systeme verwendet werden, sondern er kann auch zur präzisen Evaluation anderer Entwicklungsumgebungen eingesetzt werden. Die Struktur der Arbeit erlaubt eine inkrementelle Erweiterung und gegebenenfalls Modifizierung des Katalogs. Zum Abschluß dieses Kapitels wird die Korrelation zwischen internen und externen Qualitätsmerkmalen tabellarisch dargestellt. Diese Tabellen können direkt zur Produktevaluation benutzt werden.

Die drei zentralen internen Qualitätsmerkmale einer Entwicklungsumgebung bilden die Systemarchitektur, das Programmiermodell und die Entwicklungswerkzeuge.

In den folgenden Abschnitten werden diese Merkmale weiter präzisiert und die an sie gestellten Anforderungen zur Unterstützung externer Qualitätsmerkmale unternehmensweiter Client/Server-Anwendungen beschrieben und diskutiert.

3.2 Anforderungen an die Systemarchitektur

Die Architektur eines Systems spielt eine entscheidende Rolle bei der Unterstützung der Portabilität und Interoperabilität der mit dem System entwickelten Client/Server- Anwendungen. In Abbildung 3.1 sind die strategischen Schichten eines Systems dargestellt. Die Middleware eines Systems ist die Basisschicht, die eine Anwendung unabhängig von den benutzten Systemschnittstellen des Betriebssystems, Datenbanksystems und Kommunikationssystems macht und die für die Interoperabilität mit Fremdsoftware und externen Diensten sorgt. Realisiert wird die Basisschicht durch eine Vielzahl von Standardschnittstellen, wie Betriebssystemschnittstelle und Datenbankschnittstelle, in einer Systemarchitektur. Die Sicherung der Interoperabilität und Portabilität von Anwendungen über heterogene Plattformen hinweg gehört zu den primären Aufgaben einer Systemarchitektur. Zu

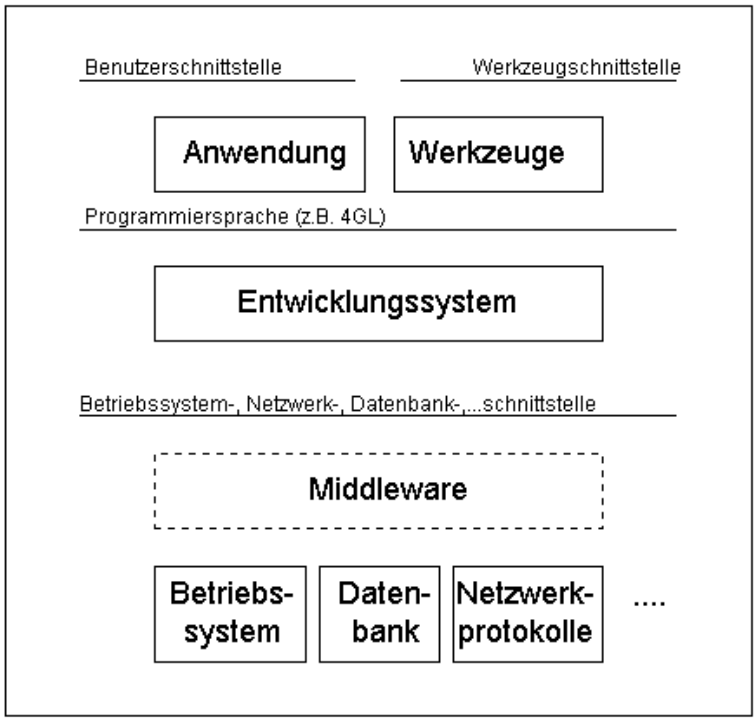

Abbildung 3.1: Die strategischen Schichten einer Softwarearchitektur

diesem Zweck sollten bei der Definition der Schnittstellen, über die Anwendungssysteme mit der Umgebung kommunizieren, Standards eingesetzt werden.

In diesem Abschnitt werden die konkreten Anforderungen an die Betriebssystemschnittstelle, Netzwerkschnittstelle, Datenbankschnittstelle und Fenstersystemschnittstelle vorgestellt. Diese Schnittstellen bilden die Ablaufumgebung einer Anwendung und können dadurch ihre Portabilität stark beeinflussen. Die offenen Schnittstellen, wie die Schnittstelle zu 3GL-Sprachen oder die Schnittstellen zu Desktop-Anwendungen, ermöglichen die Integration von neuen Anwendungen, Altsystemen und Standardsoftware. Daher sollte die Systemarchitektur einer Entwicklungsumgebung über diese Schnittstellen verfügen.

3.2.1 Betriebssystemschnittstelle

Eine Entwicklungsumgebung und deren Anwendungen werden durch die Betriebssystemschnittstelle vom hardware- und herstellerspezifischen Betriebssystem abgeschirmt. Die Landschaft unternehmensweiter Client/Server-Anwendungen kann sehr heterogen zusammengestellt sein. Dies erfordert maximale Plattformunabhängigkeit der Client- bzw. Server-Anwendungen. Zur Realisierung dieser Anforderung sollte die Betriebssystemschnittstelle folgende Leistungen gewährleisten:

- Vorhandensein des vollen Leistungsumfangs auf allen wichtigen Plattformen, z.B. UNIX-Plattformen, PC-Plattformen (DOS, Windows x.x), Macintosh und Mainframe-Plattformen.

- Modifikationen und Ergänzungen zur vorhandenen Hardware- und Softwareinfrastruktur sollen unabhängig von einem bestimmten Anbieter integrierbar sein.

3.2.2 Netzwerkschnittstelle

Ein weiteres Element der Portabilität ist der Netzwerkzugriff. Wirkungsvolle Kommunikation zwischen verschiedenen Hard- und Softwarekomponenten soll auch beim Einsatz der Produkte unterschiedlicher Hersteller gewährleistet sein. In den meisten Entwicklungsumgebungen werden die marktgängigen Kommunikationsprotokolle (z.B. TCP/IP[1], LU6.2[2]) unterstützt. Bestimmte Umgebungen bieten darüber hinaus firmeneigene Netzwerkprotokolle zur Realisierung der Kommunikation in heterogenen Umgebungen. Die Portabilität der Kommunikationsmechanismen wird aber nicht nur durch die Unterstützung marktgängiger Protokolle gewährleistet. Vielmehr kann durch das Vorhandensein einer plattformunabhängigen Netzwerk-Programmier-Schnittstelle die Portabilität unterstützt werden. Ziel ist es, eine Verteilungstransparenz zu erreichen, die das Netzwerk auf der Anwendungsebene scheinbar unsichtbar werden läßt. Auf diese Weise werden nicht nur der Anwendungsentwickler entlastet, sondern auch die Portabilität der Anwendungen durch Unabhängigkeit vom Netzwerk und der Netzwerktopologie erreicht. Bei der Realisierung einer Kommunikationsschnittstelle spielen die folgenden Anforderungen eine wesentliche Rolle:

- Der Abstraktionsgrad der Schnittstelle sollte hoch genug sein, um das Anwendungssystem frei von Spezifika des darunterliegenden Netzwerks zu halten und damit Portabilität und Unabhängigkeit sicherzustellen.

- Die Schnittstelle sollte auf allen marktgängigen Systemplattformen verfügbar und kompatibel sein.

3.2.3 Datenbanksystemschnittstelle

Eine Datenbankschnittstelle setzt die Datenbankzugriffsanweisungen eines Systemes in Anweisungen an die verwendete Datenbank um. Damit erlauben diese Schnittstellen den Datenzugriff auf verschiedene Datenbanken. Unternehmensweite Client/Server-Anwendungen verfügen über integrierte untrnehmensweite Daten, die sich auf heterogenen Datenbanksystemen befinden. Darüber hinaus können entsprechend des Wachstums des Informationsbestandes neue Datenbanken in der Umgebung eingefügt werden. Diese Eigenschaften erfordern die datenbankunabhängige Entwicklung von Client/Server-Anwendungen. Das heißt, die Client/Server-Anwendungen sollten nicht nur für die zum Entwicklungszeitpunkt bekannten und vorhandenen Datenbankmanagementsysteme entwickelt werden. Die Abbildung 3.2 stellt die Kernaufgaben einer Datenbankschnittstelle in heterogenen Umgebungen dar.

Diese Aufgaben sind:

- Unterstützung verschiedener APIs [3] zum Datenbankzugriff auf heterogene Datenbanken (relationale Datenbanken oder moderne objektorientierte Datenbanken). Für

[1] TCP/IP(Transmission Control Protocol / Internet Protocol) ist eine Protokollfamilie, die die Schichten 3 bis 7 des OSI-Referenzmodells abdeckt.

[2] LU6.2 ist das Protokoll für Advanced Program to Program Communication.

[3] API steht für Application Programming Interface.

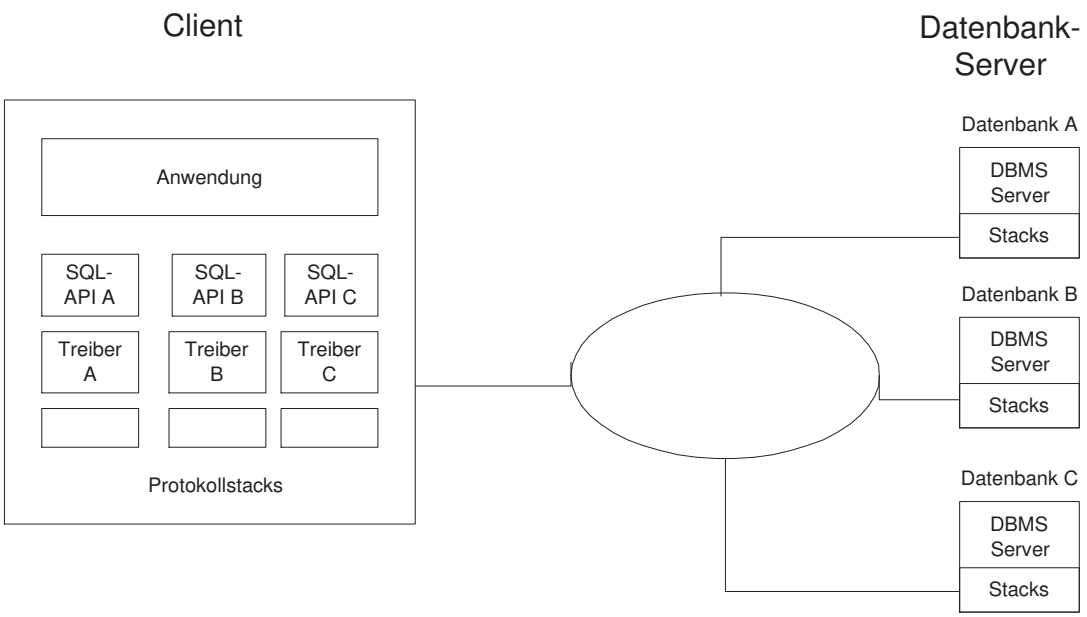

Abbildung 3.2: Eine heterogene Datenbankumgebung

ein gegebenes API (z.B. ODBC[4] kann es mehrere Datenbanktreiber (für verschiedene Produkte) und mehrere Netzwerkprotokollvarianten (z.B. TokenRing) geben, die ebenfalls unterstützt werden müssen.

- Unterstützung der entfernten Datenzugriffe auf Datenbanksysteme: Wenn es sich um eine nicht lokale Datenbank handelt, sollte die Datenbankschnittstelle einen entfernten Datenbankzugriff einleiten.

- Portabilität auf verschiedene Client-Plattformen: Beispielsweise könnten frühere Version von ODBC nur Windows und Macintosh unterstützen.

3.2.4 Schnittstellen zu externen Anwendungen

Das Hauptziel des Einsatzes von Client/Server-Lösungen für unternehmensweite Anwendungen ist die Integration unternehmensweiter Dienste. Mit Hilfe integrierter Anwendungssysteme werden unternehmensweit relevante Informationen und Dienste bereitgestellt. Dabei kann zwischen Desktop-Anwendungen (Microsoft Excel), Fremd-Anwendungen (R/3-Systeme), und Legacy-Anwendungen unterschieden werden. Die Daten- und Funktionsintegration von Client/Server-Anwendungen mit externen Dienste sollte durch das Vorhandensein folgender Schnittstellen unterstützt werden:

- Bidirektionale Schnittstelle zum Import und Export der Anwendungsdaten in bzw. aus Desktop-Anwendungen, wie z.B. OLE (Objekt Linking and Embedding) und DDE (Dynamic Data Exchange)

[4]Open Database connectivity (ODBC) ist eine offengelegte Schnittstellenspezifikation für PCs und Workstations unter Microsoft Windows (bzw. Windows NT) [Microsoft 92].

- Programmierschnittstellen für die Programm-Programm-Kommunikation mit Fremdanwendungen, wie z.B. RFC (*Remote Function Call*) für Integration der Leistung der R/3 Anwendungen.

3.2.5 Schnittstellen zu anderen Sprachen

Viele Entwicklungsumgebungen, wie Tycoon und System R/3, stellen Programmierschnittstellen für Programmiersprachen niedrigeren Abstraktionsgrads (z.B. 3GL-Sprachen) bereit. Daher ermöglicht eine bidirektionale Programmierschnittstelle zu 3GL, wie C, die Leistungserweiterung der Software durch die Einbettung von externen Subsystemen, z.B. generische Dienste, externe Bibliotheken, Fremdsoftware, und die Integrierbarkeit der Software als Subsystem in eine vorhandene Umgebung. Die Einbettung externer Subsysteme erfordert

- den Aufruf einer externen Funktion aus der Sprache,
- den Zugriff auf externen Datenstrukturen und
- die Behandlung von externen Fehlersituationen in der Sprache.

Handelt es sich bei der 3GL-Software um eine Legacy/Mainframe-Anwendung, so kann die Integration dieser Software in eine Client/Server-Umgebung erreicht werden. Ebenfalls fordert die Integration der Software als Subsystem in eine externe Umgebung

- den Aufruf der Funktionen aus der externen Sprache,
- den Zugriff auf Datenstrukturen aus der externen Sprache und
- die Behandlung von Systemfehlersituationen in der externen Sprache.

3.2.6 Fenstersystemschnittstelle

Die Grundlagen von direkt manipulativen grafischen Benutzerschnittstellen bilden Fenstersysteme *window systems* [Müßig 94]. Der Zugriff auf Fenstersysteme erfolgt über die dazugehörigen Bibliotheken, z.B. Xlib für Unix-Systeme und QuickDraw für den Macintosh sowie Win32 für Windows. Das Frontend einer Client/Server-Umgebung ist eine Sammlung heterogener Arbeitsplätze, wie z.B. Macintosh, Windows, Unix-Workstations usw. Diese Eigenschaft fordert die Unabhängigkeit der Anwendungsprogramme von Systemplattformen und ihre einfache Portierung auf verschiedene Fenstersysteme. Darüber hinaus sollen die Anwendungen sich dem Benutzer unabhängig vom Hard- und Softwarelieferanten mit gleichem *Look and Feel* präsentieren. Erreicht wird diese Portabilität durch die Spezifikation einer Schnittstelle, die zwischen dem Anwendungsprogramm und dem Zielfenstersystem arbeitet. Eine derartige Schnittstelle wandelt alle Funktionsaufrufe der systemabhängigen Programmierschnittstelle in entsprechende Funktionsaufrufe des Fenstersystems. Diese Schnittstelle kann Funktionen und Elemente aller Fenstersysteme unterstützen (Obermengenansatz) oder nur den kleinsten gemeinsamen Nenner aller Systeme darstellen (Schnittmengenansatz).

3.3 Anforderungen an das Programmiermodell

Die Programmiersprache bildet den Kernpunkt einer Entwicklungsumgebung zur Entwicklung von Anwendungslogik auf Server und Client. Jeder Programmiersprache (funktional, imperativ, objektorientiert, logikbasiert) liegt ein eigenes Denkmodell zugrunde, das den Lösungsweg mitbestimmt. So kann zum Beispiel die Verfügbarkeit oder das Fehlen einzelner Sprachelemente dazu führen, daß völlig verschiedene Entwürfe zustande kommen. Für die Entwicklung von Client/Server-Anwendungen werden in dieser Arbeit die folgenden Fähigkeiten einer Sprache in den Vordergrund gestellt:

- Es sind mächtige Datenstrukturierungskonzepte zur Verarbeitung, Auswertung und grafischen Bereitstellung der unternehmensweit integrierten Daten erforderlich.
- Strukturierungstechniken für die Programmierung im Kleinen sollte um die Aspekte der verteilten Programmierung erweiterbar sein, z.B. Behandlung von entfernt ausgelösten Ausnahmen.
- Die Sprache sollte die softwaretechnische Repräsentation der wiederverwendbaren Bausteine, z.B. durch Modularisierungskonzepte, Datenkapselung usw., unterstützen.
- Die Interoperation der über heterogene Rechner hinweg verteilten Dienste sollte durch Verteilungsabstraktion möglich sein.
- Die Dienste unternehmensweiter Client/Server-Anwendungen greifen auf persistente Massendaten zu, daher sollte die Programmiersprache Sprachelemente zur Verarbeitung persistenter Massendaten beinhalten.
- Ein Hauptbestandteil unternehmensweiter Client/Server-Anwendungen ist die grafische Benutzeroberfläche. Die Programmiersprache sollte über Sprachelemente oder Bibliotheksdienste zur Steuerung bzw. Darstellung grafischer Benutzeroberflächen verfügen.

Diese Sprachelemente und Konzepte werden im nächsten Abschnitt näher erläutert.

3.3.1 Datenstrukturierungskonzepte

Verbesserte Informationsbereitstellung durch die Integration unternehmensweiter Daten ist eine der Hauptaufgaben von Client/Server-Anwendungen. Daher spielen Datenstrukturierungskonzepte in Programmiersprachen eine tragende Rolle bei der Verwirklichung dieser Aufgaben. Zur flexiblen Modellierung der Geschäftsprozesse sollte die Sprache sowohl über vordefinierte Datentypen und dazugehörige Operationen als auch über Konzepte zur Unterstützung benutzerdefinierter Datentypen verfügen. Diese Daten sollten auch plattformunabhängig und typsicher zwischen verteilten Diensten austauschbar sein.

In diesem Abschnitt werden die unter Datenstrukturierungskonzepten gruppierten internen Qualitätsmerkmale diskutiert. Dies sind die Grundkonzepte zur Darstellung der Daten in Programmiersprachen, vordefinierte Datentypen, die Typkonstruktoren und benutzerdefinierte Datentypen.

Vordefinierte (atomare) Datentypen

Zu jeder Sprache gehört ein Satz vordefinierter Datentypen, wie z.B. *integer, real, char, string* usw. Diese Datentypen können mit einer Reihe vordefinierter Operationen wie z.B. den arithmetischen Operationen manipuliert werden, deren Implementierungsdetails dem Benutzer verborgen bleiben. Die Verwendung dieser Datentypen wird durch die vorher festgelegte Semantik spezifiziert, die entweder explizit in der Sprachdefinition angegeben wird oder die implizit bekannt ist. Die Portabilität der vordefinierten Datentypen spielt bei der Entwicklung datenintensiver Client/Server-Anwendungen eine wichtige Rolle. Dies ermöglicht die flexible Verteilung einer Anwendung auf unterschiedliche Plattformen. Die Portabilität soll zusätzlich durch die automatische Konvertierung der Darstellung der Datentypen während der Verteilung der Komponenten auf unterschiedliche Plattformen unterstützt werden. Die Konvertierung setzt das Vorhandensein einer Darstellung des Datentyps auf allen Plattformen voraus.

Einen Hauptbestandteil einer Client/Server-Umgebung bilden Relationale Datenbanken. In diesem Datenbankmodell sind Datentypen zur flexiblen Behandlung von Textdaten, Datums- bzw. Zeit- und Nullwerten vorgesehen. Daher sollte die Programmiersprache Konzepte zur Unterstützung dieser Datentypen bieten.

- Vollständige Darstellung von Zeichenketten: Textdaten bilden einen großen Teil der Information einer betrieblichen datenintensiven Anwendung. Der flexible Umgang mit Textdaten sollte durch global verfügbare Operationen, wie z.B. das Verknüpfen mehrerer Zeichenketten, erleichtert werden. Darstellung von Texten in Sprachen mit großem Alphabet (Japanisch, Hebräisch) sollte durch einen *Double Byte Character* Zeichensatz unterstützt werden.

- Datums- und Zeitfunktionen: Datumsberechnungen werden häufig von betrieblichen Anwendungen benutzt. Diese Operationen sind durch spezielle Funktionen in der Programmiersprache zu unterstützen. Die Berechnung von Datums- und Zeitintervallen, die Addition bzw. Subtraktion einer Zahl zum/vom Datum, das Extrahieren eines Teils eines Datums (Monat, Jahr oder Tag) können als Grundoperationen für die Datentypen Datum und Zeit angesehen werden.

- Plattformunabhängige Genauigkeit der Finanzarithmetik für betriebliche Daten sollte vorhanden sein. Insbesondere sind Festkommazahlen zusätzlich zu Fließkommazahlen zu unterstützen.

- Wenn die Programmiersprache Datentypen unterstützt, die nicht von Datenbanken unterstützt werden (z.B. Aufzählungstypen), sind bidirektionale Typumwandlungsoperationen vorzusehen.

- Unterstützung der Typumwandlung: Die kontrollierte Typumwandlung für alle vordefinierten Datentypen erhöht die Flexibilität der Darstellungsformen der Datentypen wie Datum, Zeit und Zahl als Text, z.B. auf grafischen Oberflächen, und erleichtert die Durchführung der Operationen wie Datumsberechnung der Form Datum = Datum + ganze Zahl. Die Datentypen Zeichen, Datum, Zeit und Zahl werden in datenintensiven Anwendungen benutzt, diesbezügliche Typumwandlungen sollten mindestens vorhanden sein.

- Explizite und konsistente Unterstützung von ausgezeichneten Nullwerten: Beim Datenaustausch zwischen Datenbanken und Programmiersprachen sollen auch Null-

werte, die vom Datenbankmodell definiert sind, mit wenig Aufwand von Anwendungsprogrammen angenommen und auf grafischen Benutzeroberflächen dargestellt werden können.

Typkonstruktoren

Wenn komplexe Daten zu verarbeiten sind, spielt die Verfügbarkeit von Typkonstruktoren in der Programmiersprache eine wichtige Rolle. Typkonstruktoren erlauben die präzise Modellierung von komplexen Informationsstrukturen. Bietet eine Sprache einen Satz von Konstruktoren, kann der Benutzer durch geschachtelte und rekursive Anwendung dieser primitiven Typkonstruktoren zu problemadäquaten Informationsstrukturen gelangen (z.B. dynamische Felder oder statische Felder). Von wesentlicher Bedeutung in der Programmierung datenintensiver und langlebiger Anwendungen ist das Konzept der Aggregation [Hoare 68] und [Smith 77]. Mit diesem Konzept können mengenwertige Datentypen zur Aufnahme komplexer, umfangreicher Zwischenergebnisse dargestellt werden. Die Aufnahme umfangreicher Zwischenergebnisse ist die Aufgabe der Anwendungslogik einer Client/Server-Anwendung. Daher sollte die Programmiersprache entsprechende Datentypen zur Aufnahme dieser Daten besitzen. Records und Arrays bieten Mechanismen zur Aggregation der Daten und deren Kapselung. Zur Darstellung komplexerer Strukturen sollten diese Typkonstruktoren folgende Merkmale besitzen:

- Die Sichtbarkeitsbereiche der Feldnamen sollten auf den Recordkonstruktor beschränkt sein.
- Wünschenswert sind Mechanismen zur dynamischen Record- und Arrayerweiterung.
- Neben dem Zugriff auf einzelne Recordfelder bzw. Arrayfelder (Feldselektion) ist der Zugriff auf Records /Arrays als Ganzes zu unterstützen. Einige Sprachen ermöglichen auch die automatische Projektion von Records/Arrays auf Teillisten.
- Ein Record mit mehr Feldern kann als Wert eines Recordtyps mit weniger Feldern betrachtet werden. Solche Spezialisierungsbeziehungen zwischen Recordtypen sollten durch Anwendungsprogramme genutzt werden können.
- Programmierer sollten von der Repräsentation von Records und Arrays im Speicher abstrahieren können. Dementsprechend sollten keine Einschränkungen bezüglich der Maximalgröße von Records existieren.
- Verschiedene Typsichten auf dieselbe Recordvariable sollten darstellbar sein (z.B. implizit durch Overlay-Anweisungen oder explizit durch variante Recordtypen).

Benutzerdefinierte Datentypen

Eine Eigenschaft moderner Programmiersprachen ist die zusätzliche Möglichkeit der Definition von Datentypen durch den Benutzer. So kann z.B. der Benutzer numerischer Bruch als Datentyp definieren. Zur Ausführung von Operationen auf den Datentypen soll der Benutzer wiederverwendbaren Code schreiben können. Durch einfache Funktionsdefinitionen, die in den meisten Programmiersprachen verfügbar sind, werden diese benutzerdefinierten Funktionen jedoch nicht direkt mit den Datentyp assoziiert. Außerdem sind die Implementationsdetails des Datentyps und der Operationen im gesamten Programm sichtbar. Kapselung der Datenzugriffe kann durch abstrakte Datentypen (ADT) unterstützt werden.

Ein ADT besteht aus einem Datentyp und einer Menge auf diesem Datentyp definierter Operationen. Ausschließlich diese Operationen sind zur Manipulation der Werte des ADT möglich, wobei für den Benutzer die Implementierung des Datentyps und der Operationen nicht sichtbar ist. Die in einer unternehmensweiten Client/Server-Anwendung definierten ADTs sollen

- über das Netz austauschbar und
- persistent in einer Datenbank speicherbar sein.

Weiterführende Typkonzepte

Typen gehören zu den wichtigsten Modellierungsmitteln der praktischen Informatik. Sie erlauben eine Beschreibung allgemeiner Eingabeklassen, Ausgabeklassen und Zustandsklassen von Algorithmen. Typinformationen unterstützen die Effizienz, Verständlichkeit und Korrektheit von Algorithmen und Datenstrukturen. Typsysteme moderner Sprachen verfügen über weitere Merkmale, die typsichere Definitionen generischer, erweiterbarer Dienste ermöglichen. Diese Merkmale sind:

- Typisierung: Dezentrale Client/Server-Programmierung sollte sich auf eine gemeinsame Notation zur Typisierung beim Datenaustausch verlassen können. Um flexibel verteilte Programmierung zu ermöglichen, sollten die Daten über wohldefinierte typisierte Bindungen ausgetauscht werden. Darüber hinaus sollte der Typ einer Netzwerkverbindung im Typsystem der Programmiersprache abgebildet werden können [Cardelli 97].
- Strukturäquivalenz von Typen: Zur Festlegung der Äquivalenz zweier Typen können ihre Namen oder ihre Strukturen verglichen werden. Strukturäquivalenz von Typen kann durch Typregeln genau und einfach definiert werden. Dies erlaubt den typsicheren Datenaustausch. Im Gegensatz dazu können durch Namensäquivalenz, die die Definition von Typen mit gleichem Namen am selben Ort voraussetzt, die im Raum und über die Zeit verteilt definierten und übersetzten Programmressourcen nicht adäquat behandelt werden.
- Typüberprüfung: Die Sicherheit einer Programmiersprache ist im wesentlichen durch ihre Mechanismen zur Typüberprüfung (auch über Modulgrenzen hinweg) bestimmt. Typüberprüfungen verhindern, daß Operationen auf Werte angewendet werden, die auf ihnen nicht definiert sind. Typüberprüfungen müssen vor der eigentlichen Operation, also entweder schon zur Übersetzungszeit oder spätestens direkt vor der Operation zur Laufzeit stattfinden. Typüberprüfungen zur Laufzeit verringern die Effizienz und führen zu einem erhöhten Speicherbedarf, da die Typen an die Werte gebunden werden müssen. Für die Entwicklung von Client/Server-Anwendungen ist eine Sprache mit statischer Typisierung vorzuziehen. Statische Typisierung bedeutet, daß der Compiler für jeden Ausdruck feststellen kann, welchen Typ er zur Laufzeit haben wird. Typfehler werden bei der Übersetzung erkannt und können somit zur Laufzeit nicht auftreten. Statische Typüberprüfung schließt allerdings den dynamischen Typtest nicht aus. Es existieren Situationen, in denen es erforderlich ist, die Typüberprüfung auf wohldefinierte Zeitpunkte zur Programmevaluation zu verzögern. Typische Beispiele sind die Parameterübergabe beim entfernten Prozeduraufruf oder Elementselektion aus heterogenen Mengen. In diesen Beispielen existiert

in dem Kontext, in dem ein Wert erzeugt wurde, und in dem Kontext, in dem er benutzt wird, keine gemeinsame, statisch überprüfbar Spezifikation. [Matthes 93].

- Polymorphismus: Polymorphismus liegt vor, wenn eine Programmvariable zur Laufzeit Ausprägungen verschiedener Datentypen annehmen kann. Während in monomorphen Programmiersprachen Funktionen bzw. Prozeduren und ihre Parameter, Operatoren und deren Operanden etc. einen eindeutigen Typ besitzen, erlauben polymorphe Sprachen dafür mehr als einen Typ [Watt 90]. Polymorphismus ist von besonderer Bedeutung für die Wiederverwendbarkeit und Erweiterbarkeit der Software in großen Anwendungen. Viele Standardalgorithmen und Datenstrukturen sind unabhängig vom Typ der Variablen und können durch polymorphe Funktionen bzw. polymorphe Typkonstruktoren realisiert werden. Neben *Ad hoc* Polymorphismus [5] und parametrischer Polymorphismus [6] ist Subtyppolymorphismus [Cardelli, Wegner 85] ein wichtiges Mittel für die inkrementelle Softwareentwicklung und die Erweiterung der Funktionalität bestehender Software . Algorithmen, die für einen bestimmten Typ T entworfen sind, können auch für die Werte eines Subtyps von T, der mit weiteren Eigenschaften erweitert ist, benutzt werden.

- Typvollständigkeit: Eine Sprache, in der die Typoperatoren ohne Einschränkung geschachtelt werden dürfen, heißt typvollständig [Watt 90]. Typvollständigkeit ist eine wichtige Forderung an anwendungsnahe Sprachen, da nur so eine vollständige Abstraktion von den Details einer komplexen Informationsstruktur möglich ist. Darüber hinaus fordert man Typorthogonalität, d.h. die übrigen statischen und dynamischen semantischen Regeln der Sprache gelten uniform für die Werte aller Typen.

3.3.2 Strukturierungstechniken für die Programmierung im Kleinen

In diesem Abschnitt werden die Möglichkeiten einer Sprache zur Beschreibung von Algorithmen dargestellt. Hierbei werden die blockstrukturierten Kontrollstrukturen, prozedurale Abstraktion, Ausnahmebehandlung, Multi-Threading, Transaktionen und Sperrkonzepte einer Sprache vorgestellt. Algorithmen zur Entwicklung unternehmensweiter Client-/Server-Anwendungen unterscheiden sich von Algorithmen für lokale, monolithische Anwendungen. In einer Client/Server-Anwendung bilden Abstraktionen die kleinsten abgeschlossenen Einheiten, die einen Dienstnehmer (Client) oder Diensterbringer (Server) realisieren. Die Ausnahmen werden nicht nur lokal, sondern auch entfernt ausgelöst. Die Transaktionen können sich auf mehrere Abstraktionen beziehen. Diese Eigenschaften der Algorithmen für Client/Server-Anwendungen erfordern besondere Techniken für die Programmierung im Kleinen, die in diesem Abschnitt diskutiert werden.

Blockstrukturierte Kontrollstrukturen

Das Vorhandensein von Sprachelementen zur Strukturierung der Ablaufsteuerung ist ein wichtiges Qualitätsmerkmal von Sprachen für imperative Programmierung. Client/Server-Anwendungen legen imperativ zustandsbasierte, transaktionale Algorithmen zugrunde. In diesen Algorithmen werden destruktive Zuweisungen zur Zustandsänderung, Schleifen zur Realisierung von zyklischen Wiederholungen und bedingte Ausdrücke zur Darstellung

[5] Ad-hoc Polymorphismus liegt vor, wenn z.B. eine Funktion auf einer endlichen Menge von Typen in einer vom Typ abhängigen Weise operieren kann.

[6] Parametrischer Polymorphismus ermöglicht die Generalisierung.

von Verzweigungen eingesetzt. Darüber hinaus werden die Algorithmen der komplexen Client/Server-Anwendungen überschaubarer und können leichter verifiziert, geändert und erweitert werden. Böhm und Jacopini [Böhm, Jacopini 66] haben gezeigt, daß jeder Algorithmus durch Kombination der Bausteine Sequenz, Verzweigung und Wiederholung, die alle nur einen Eingang und einen Ausgang haben, darstellbar ist.

Prozedurale Abstraktion

Abstraktion erlaubt die Trennung zwischen der Spezifikation und der Implementation eines Programmfragments. Diese Trennung ermöglicht eine Veränderung der Implementation ohne Konsequenzen für die Programme, die sie benutzen. Bei der Programmierung im Kleinen wird zwischen Funktionsabstraktion und Prozedurabstraktion unterschieden. Eine Funktionsabstraktion ist eine Abstraktion über einen Ausdruck, der ausgewertet einen Wert ergibt [Watt 90]. Eine Prozedurabstraktion abstrahiert von einem Kommando, das ausgeführt Variablen verändert. Der effiziente Einsatz dieser Abstraktionsmechanismen erfordert

- Benennung: Sichtbarkeitsregeln für Namen. Statische oder dynamische Benennung.
- Parametrisierung: Um die Vorteile der Abstraktion zu vervollständigen, sollte die Paramertisierung der Abstraktion in Bezug auf ihre Variablen möglich sein.
- Unterstützung der Korrespondenzprinzips: Es sollte für jede Form der Datendeklaration ein entsprechender Parameter existieren und umgekehrt.
- Lokalität: Ein Programmfragment sollte auch lokal benannte Objekte besitzen können (lokale Variablen, Typen und Prozeduren).

Ausnahmebehandlung

Das Verhalten von Programmen in Fehlersituationen oder beim Auftauchen unerwarteter Ereignisse wird durch den Mechanismus der Ausnahmebehandlung gesteuert. In unternehmensweiten Client/Server-Anwendungen können Ausnahmen in entfernten Prozeduren oder in entfernten Datenbanken als Folge der Integritätsverletzung entstehen. Die flexible Behandlung dieser Ausnahmen erfordert

- das transparente Abfangen der in entfernten Prozeduren ausgelösten Ausnahmen,
- die Behandlung der Ausnahmen, die von externen Diensten(z.B. Funktionen integrierter Systeme) ausgelöst werden,
- die Abbildung der vom Datenbanksystem ausgelösten Ausnahmen in Ausnahmen der Sprache.

Multi-Threading

Threads, die auch *lightweight processes* [Cardelli et al. 88], [Keppel 93]genannt werden, sind Einheiten der Nebenläufigkeit innerhalb eines Programmes. Sie werden insbesondere zur Realisierung nebenläufige ereignisorientierte Server-Programme, die mehrere Clients parallel zu bedienen haben, benutzt. Jedem wartenden Ereignis kann ein Thread zugeteilt werden, der auf das Auftreten des Ereignisses wartet. In dieser Zeit können andere

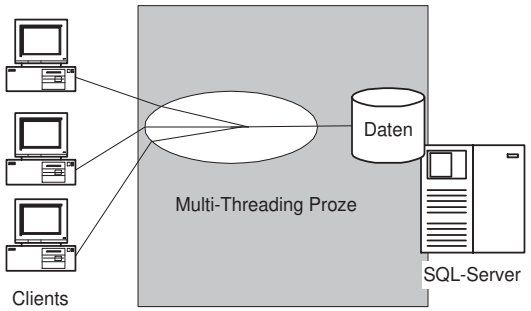

Abbildung 3.3: Multi-Threading Architektur für Server

Threads aktiv sein. Unter Multi-Threading wird die Möglichkeit verstanden, innerhalb eines durch eine Programmaktivierung entstandenen Betriebssystemprozesses gleichzeitig mehrere nebenläufige Threads ablaufen zu lassen. Dies ermöglicht dem Server die parallele Verarbeitung der Anfragen mehrerer Clients durch das Erzeugen von Server-Threads. Eine Multi-Threading Architektur für Server, wie in Abbildung 3.3 liefert die beste Performanz durch das Ausführen von Benutzerverbindungen, Anwendungen und Datenbankoperationen im selben Adreßraum.

Diese Architektur besitzt einen eigenen Scheduler und ist dadurch unabhängig von Tasking und Adreßschutzmechanismen des Betriebssystems. Dies erhöht die Plattformunabhängigkeit der Server-Implementierung, da der Server viele lokale Dienste des Betriebsystems nicht mehr braucht. [Orfali et al. 96]

Persistente Threads

In den meisten Programmiersystemen sind Threads flüchtige Objekte. Ihre Lebensdauer ist durch den sie umgebenden Betriebssystemprozeß begrenzt. Um nach einem Programmneustart einen bestimmten Evaluationszustand automatisch wiederherstellen zu können, ist besonders bei komplexeren Anwendungen ein hoher Aufwand erforderlich. Dieser Aufwand und die damit verbundenen Komplikationen lassen sich durch die persistente Speicherung von Threads reduzieren [Mathiske 96]. Das Speichern der Evaluationszustände und ihre Wiederherstellung erlauben die Realisierung langlebiger, verteilter Aktivitäten, die eine zentrale Rolle in unternehmensweiten Client/Server-Anwendungen spielen. In [Matthes, Schmidt 94] wird aufgezeigt, wie die Realisierung verteilter Arbeitsabläufe mit Hilfe persistenter Threads vereinfacht werden kann.

3.3.3 Strukturierungskonzepte für die Programmierung im Großen

Grundvoraussetzung für das Erstellen von Softwaresystemen mit einem hohen Maß an Erweiterungsfähigkeit, Wiederverwendbarkeit und Kompatibilität ist unter anderem eine modulare Struktur der Software [Meyer 90].

Client/Server-Anwendungen weisen eine erhebliche Komplexität auf. Dies erfordert leistungsfähige Konzepte zur Modularisierung des Quellcodes und zur Organisation der Module.

Modularisierungskonzepte

Modularisierung heißt, bestimmte Verarbeitungsfolgen physisch aus einem Programm auszulagern. Dies kann beispielsweise durch eine Trennung von Bildschirmbearbeitung und Datenbankzugriff erfolgen und gewinnt an Bedeutung mit der Einführung von Client/Server-Modellen. Client/Server-Computing ist die logische Erweiterung der modularen Programmierung. Die Unterteilung monolithischer Software in Module hat eine einfachere Entwicklung und effizientere Wartung der Software zur Folge. Beim Client/Server-Computing müssen diese Module nicht alle im gleichen Adreßraum ablaufen. Bei dieser Architektur erhalten die aufrufenden Module die Rolle des Clients, der einen Dienst anfordert, und die aufgerufenen Module werden als Server, die die Dienste erbringen, betrachtet. Damit ist Modularisierung bei der Gestaltung von Client/Server-Anwendungen eine technische Notwendigkeit. Im Rahmen der Einführung einer Client/Server-Architektur ist eine korrekte Modularisierung nicht nur Voraussetzung für effiziente Systeme, sondern ermöglicht auch die Beseitigung von Redundanz durch Wiederverwendung von Modulen. Goos und Kastens [Goos, Kastens] geben eine Definition des Modulbegriffs an, die folgende Anforderungen umfaßt:

- Die Kommunikation eines Moduls mit der Außenwelt soll nur über eine eindeutig spezifizierte Schnittstelle erfolgen.
- Zur Integration eines Moduls in ein Programmsystem darf keine Kenntnis seines inneren Arbeitens erforderlich sein.
- Ein Modul soll zusammengehörige Funktionen gemeinsam einschließen.
- Ein Modul soll getrennt übersetzbar sein.

Zusätzlich sollten die folgenden Kriterien berücksichtigt werden:

- Das Modul soll in einer eigenen Testumgebung geprüft werden können.
- Module müssen in Bibliotheken verwaltet werden können.
- Die Ausnahmebehandlung muß besonders gesteuert werden können.
- Module sollen über Adreßraumgrenzen hinweg aufgerufen werden können.

Zusammengefaßt sollten Module Einheiten darstellen, die potentiell über die Grenzen von Anwendungen, Sitzungen, Systemen und Plattformen hinaus, also global, aufbewahrt, verteilt und verwendet werden können. Andererseits können sie zentral definiert, implementiert und gewartet werden.

Wiederverwendung von Code

Das Komplexitätsproblem der unternehmensweiten Client/Server-Anwendungen kann durch Wiederverwendung existierender Komponenten vermindert werden. Darüber hinaus gewährleistet der Einsatz wiederverwendbarer Einzelbausteine eine redundanzfreie Programmierung und reduziert damit den Wartungs- und Pflegeaufwand. Die Wiederverwendung bei der konventionellen Programmentwicklung ist auf Aufrufe von Prozeduren in vorgefertigten oder selbstangelegten Bibliotheken eingeschränkt. Bei modernen Programmiermodellen kann die Wiederverwendung durch gezieltes Erben und Anpassen bestehender Komponenten deutlich verbessert werden. Grundsätzlich bieten sich neben oben beschriebenen

Modularisierungskonzept die folgenden Mechanismen zur Wiederverwendung von Code an:

- Copycode, Include-files: In vielen Sprachen, wie NATURAL und ABAP/4, werden Programmteile, wie z.B. eine Sammlung von Unterprogrammen oder Datendeklarationen, zur Wiederverwendung angeboten. Diese Komponenten können innerhalb eines Programms referenziert werden. Diese Referenzen haben eine Kopiersemantik, d.h. zur Übersetzungszeit wird eine Kopie der referenzierten Komponente erzeugt. Die Modifizierung dieser Programmteile hat die erneuete Übersetzung der Wirtsprogramme zur Folge.

- Vererbung von Code: Wiederverwendung ist auch ein Aspekt der Vererbung in objektorientierten Sprachen. Unter Vererbung ist jener Mechanismus zu verstehen, der es ermöglicht, Attribute und Methoden von Klassen an andere Klassen weiterzugeben. Eine vererbende Klasse (Superklasse) gibt Attribute und Methoden an eine erbende Klasse (Subklasse) weiter, welche von dieser erweitert (um zusätzliche Attribute und Methoden) bzw. modifiziert (Redefinition von Methoden) werden können. Wenn die Eigenschaften unverändert übernommen werden, spart der Programmierer nicht nur Zeilen im Code, sondern auch Zeit zur Entwicklung, Dokumentation und zum Testen dieses Codes.

- Datenkapselung: Datenkapselung geht auf das Konzept des abstrakten Datentyps zurück, in dem eine Datenstruktur mit den zugehörigen Operationen zusammengefaßt wird. Damit sollen die Eigenschaften einer guten Modularisierung erreicht werden. Dahinter steht der Gedanke, Module mit starker interner Bindung und geringer externer Kopplung zu schaffen [Denert 91]. Außerhalb des abstrakten Datentyps ist nur dessen Verhalten bekannt, nicht aber die Art der Implementierung. Gleiche Anforderungen werden an die wiederverwendbaren Dienste in einer Client/Server-Architektur gestellt. Daher kann ein abstrakter Datentyp als die beste softwaretechnische Repräsentation eines Dienstes in der Client/Server-Anwendung angesehen werden.

Die Wiederverwendung von Code kann durch eine oder mehrere der oben erwähnten Methoden unterstützt werden. Es sollte allerdings zur Reduktion der Komplexität der Programmumgebung nur eine der o.g. Methoden verwendet werden.

3.3.4 Verteilungsabstraktion

Lokale Softwarearchitekturen können relativ einfach zu verteilten Softwarearchitekturen skaliert werden. Wie es in Abbildung 3.4 dargestellt ist, bilden Daten, Algorithmen und Prozesse die drei wesentlichen verteilbaren Komponenten einer Software.

Daher können folgende Ansätze zur Verteilungsabstraktion unterschieden werden:

- entfernter Datenbankzugriff

- entfernter Prozeduraufruf

- Mobilität von Code

- Mobilität von Prozessen

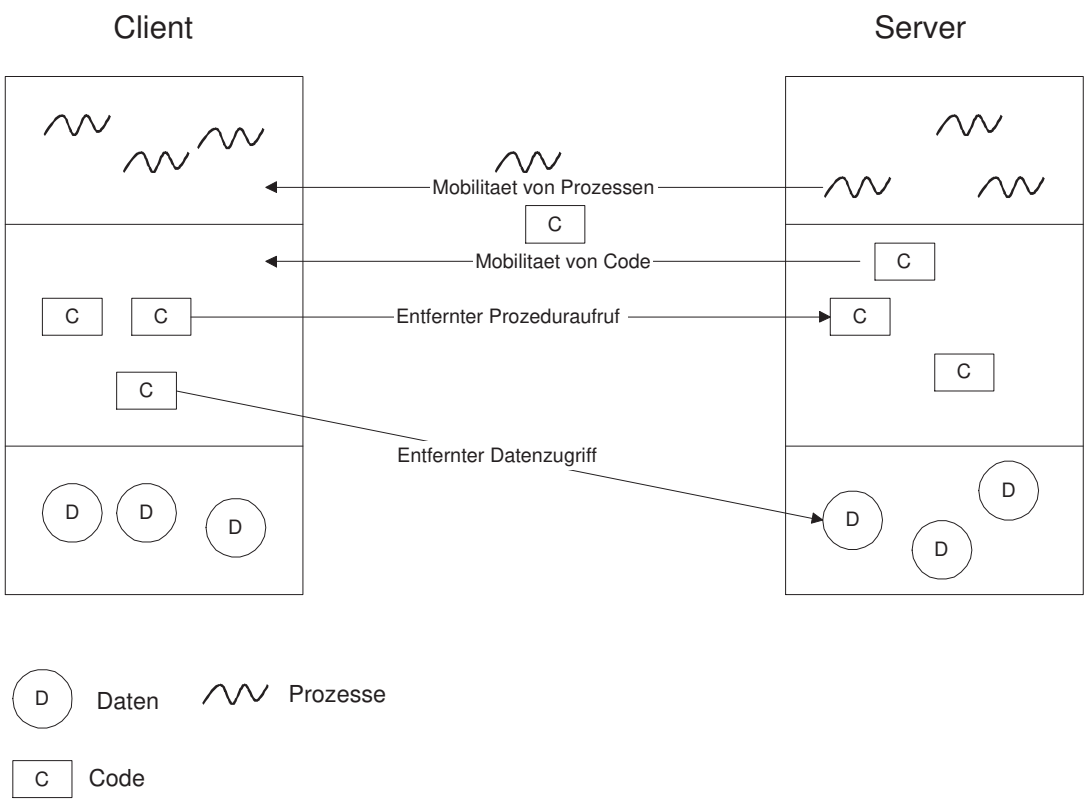

Abbildung 3.4: Verteilungsabstraktion

Durch diese Abstraktionsmechanismen sollten die komplizierten Aspekte einer Netzwerkkommunikation auf logisch höheren Ebene vor dem Anwendungsprogrammierer verborgen werden. Daher ist ein generelles Entwurfsziel bei der Realisierung dieser Ansätze zur Verteilungsabstraktion die Forderung nach der Unsichtbarkeit des verteilten Charakters des Systems für den Benutzer. Walker und Popek unterscheiden [Walker, Popek 89] die folgenden Aspekte des Transparenzbegriffs:

- Ortstransparenz: Der Benutzer benötigt kein Wissen darüber, wo die Verarbeitungsleistung oder auch Teile davon erbracht werden.
- Performanztransparenz: Der zusätzliche Zeitaufwand, der durch den verteilten Charakter eines Systems entsteht, soll so klein sein, daß er von einem Benutzer nicht wahrgenommen werden kann.
- Zugriffstransparenz: Sämtliche Systemressourcen sind über einheitliche Zugriffsfunktionen unabhängig von ihrer Lokation ansprechbar.

Weitere für alle drei Ansätze relevante Entwurfsziele sind

- Unterstützung der Sicherheitsaspekte: Es sollten Konzepte zur gegenseitigen Authentisierung von Client und Server vorhanden sein.
- Heterogenität: Die Protokolle zur Verteilungsabstraktion sollten der Anwendung erforderliche Konvertierungsmechanismen für Parameter zwischen verschiedenen Datenformaten (z.B. Strings in ASCII- bzw. EBCDIC-Format) anbieten.
- Skalierbarkeit: Es sollte leicht möglich sein, neue Middlewareanbieter aufzunehmen.

Nachfolgend werden vier Ansätze der Verteilungsabstraktion beschrieben und ihre individuellen Eigenschaften und relevanten Anforderungen diskutiert.

Entfernter Datenzugriff

Mechanismen zum Datenbankfernzugriff in offenen Netzen ermöglichen Client-Anwendungen den Zugriff auf entfernte (persistente) Daten auf dem Server. Die Datenbankschnittstellen haben die Aufgabe der Weiterleitung der Datenzugriffsanweisungen des Systems an die Datenbank. Wenn es sich um einen Datenzugriff auf eine nicht lokale Datenbank handelt, werden diese Anweisungen über das Netz gesendet, um dem Client einen entfernten Datenzugriff auf den Server zu ermöglichen. Beispielsweise ermöglicht Remote SQL entfernten Datenzugriff auf relationale Datenbank in verteilten Client/Server-Umgebungen. Die Datenbankzugriffe werden vom Datenbanksystem völlig transparent für die Anwendung vom Client zum Server übertragen. Datenbankanfragen bestehen (zur der Zeit) zum großen Teil aus SQL-Anweisungen. Wenn diese Anweisungen nacheinander an den Server gesendet werden, kann es sehr schnell zu einer hohen Belastung des Netzes führen. Ebenfalls kann das Ergebnis der Anfragen sich um große Tabellen handeln, die ebenfalls einen hohen Netzverkehr verursachen können [Potter, Dr. Cory 93].

Entfernter Prozeduraufruf

Der entfernte Prozeduraufruf (engl. Remote Procedure Call, kurz RPC) erweitert den von Standardprogrammiersprachen her bekannten Aufrufmechanismus in der Form, daß Prozeduraufrufe auch über Rechnergrenzen hinweg möglich sind. Er stellt die eigentliche

Kommunikationskomponente zwischen dem Client- und dem Server-Prozeß dar. Seine Aufgabe ist es, die notwendige Kommunikation vor dem Anwendungsprogramm zu verbergen, so daß dieses sich idealerweise nicht von einer lokalen Variante unterscheidet [Johannisson 95]. Da es zur Zeit noch keinen Standard für RPC gibt, realisieren der Entwicklungsumgebung eigene RPC-Mechanismen. Hier sind einige wichtige Eigenschaften zusammengefaßt, die die Leistungsfähigkeit des RPC beeinflussen:

- Parameterstrukturen: Die übertragenen Aufruf- und Ergebnisparameter sollten von beliebig komplexer Struktur sein. Idealerweise sollte auch die Übergabe von Prozeduren und Funktionen als Parameter möglich sein. Beispielsweise sollten polymorphe Funktionen sowie Funktionen höherer Ordnung definiert werden dürfen.

- Aufrufsemantik: Die Aufrufsemantik eines RPC beschreibt, welche Garantien bzgl. der Aufrufdurchführung dem Client bei bestimmten Fehlerfällen gemacht werden. Für den Regelfall sollte die Semantik *at-most-once*[7] unterstützt werden. Dies bedeutet, daß ein RPC genau einmal durchgeführt wird, wenn der Kommunikationskanal verfügbar ist. Bei dauerhaften Fehlern sollte der Client eine explizite Fehlermeldung erhalten [Mathiske 96].

- Mögliche Aufrufstrukturen: RPCs sollten beliebig geschachtelt aufgerufen werden können. Außerdem sollte der Rückaufruf eines Servers an einen Client ohne Verklemmung möglich sein.

- Arten des Datentransfers: Im Rahmen von RPCs werden Daten normalerweise als Wertparameter übergeben. Bei vielen Systemen wird ein lokaler Funktionsaufruf durch Übergabe vom Referenzparametern realisiert. Es sollte von diesem Unterschied auf Benutzerebene abstrahiert werden. Damit können entfernte und lokale Aufrufe einheitlich programmiert werden.

- Massendatentransfer: Für große Datenmengen (z.B. Relationen, Tabellen) sollten spezielle Übertragungstechniken vorhanden sein.

Mobilität von Code

Mobilität von Code ist ein moderner Ansatz, der wie RPC von verteiltem Code abstrahiert. Im Gegensatz zum RPC-Mechanismus, in dem die aufgerufende Funktion auf dem Server ausgeführt wird und nur die Ergebnisse zur Dienstfordernden (Client) über das Netz zurückgesendet werden, migriet die aufgerufene Funktion zum Client und wird auf dem Client ausgeführt. Die Tatsache, daß der mobile Code auf dem Client-Rechner ausgeführt wird und damit auf lokale Ressourcen und den Arbeitsspeicher zugreifen kann, erfordert vor allem die Portabilität von Code. Weiterhin werden folgende Dienste zur Begleitung der Migration von Code und seiner Ausführung auf dem Client-Rechner erforderlich [Orfali et al. 96]:

- Eine sichere Umgebung für die Ausführung von Code: Es sollten Mechanismen zum Kontrollieren der Ausführungsumgebung, d.h. Zugriff auf den Arbeitsspeicher, Systemaufrufe und die Kommunikation mit dem Server, vorgesehen sein.

[7] Weitere Möglichkeiten sind: *maybe, at-least-once* und *exactly-once*.

- Plattformunabhängige Dienste: Das System sollte über Dienste plattformübergreifenden Verwaltung von Arbeitsspeicher, Threads, Synchronisations- und Kommunikationsmechanismen verfügen. Der Code sollte auf verschiedenen Betriebssystemen und Plattformen ausführbar sein.
- Kontrollierter Lebenszyklus: Der Client sollte über eine Umgebung für loading, unloading und Ausführung von Code verfügen.
- Verteilungsmechanismen: Es sind Mechanismen zu sicheren Verteilung von Code im Netz erforderlich. Zum Beispiel sollte die Identität des Codes auf dem Server und auf dem Client bestätigt werden.

Mobilität von Prozessen

Migrierende Prozesse sind ein idealer Ansatz zur Realisierung eines aktivitätsorientierten Stils der Modellierung verteilter kooperativer Informationssysteme, der von Skripten in Taxis [Borgida et al. 93], prozeßorientierten Spezifiktionen in Estelle [Turner 93] oder von (visuellen) Prozeßsprachen von Workflow-Management-Tools wie z.B. Regatta [Swenson 93] bekannt ist [Mathiske 96]. In einer verteilten aktivitätsorientierten Anwendung (z. B. Workflow) erfolgt die Ausführung eines komplexen Ablaufes beginnend auf einem Rechner über mehrere Rechner hinweg. Es existieren verschiedene Ansätze zur Modellierung migrierender Aktivitäten, wie z.B. verteilte Objektsysteme mit entfernten Referenzen, Versenden der Aktivitäten im RPC-Ansatz. Daher ist Mobilität von Prozessen im wesentlichen eine Konsequenz der Anforderungen an z.B. einen RPC-Mechanismus. Trotzdem sollten speziell für Prozesse erforderlichen Mechanismen konkretisiert werden. Erforderlich sind bei dieser Erweiterung folgende Charakteristika:

- Die Prozesse sollen nicht nur passiv nach Anfrage als RPC-Parameter migrieren sondern vielmehr selbst ihre eigene Migration veranlassen.
- Die mobilen Prozesse sollen in der Lage sein, dynamische Bindungen an jeweilige lokale Objekte einzugehen und wieder zu lösen, um lokale Ressourcen zu nutzen und mit anderen Aktivitäten zu kooperieren [Mathiske 96].
- Prozesse sollen komplett migrieren, damit sie unabhängig und autonom ausführbar sein können.

Dieser Ansatz ist flexibler als Mobilität von Code, da migrierende Prozesse keine weitere Kommunikation mit dem Server benötigen und ihre Laufzeitumgebung mitbringen. Diese Umgebung kann allerdings für GUI-erhaltende Prozesse sehr groß werden. Mit flexiblen Bindungsmechanismen für ubiquitäre Ressourcen und Replikation von Code kann die Größe des Codes erheblich reduziert werden [Mathiske 96].

3.3.5 Persistenzabstraktion

In unternehmensweiten Client/Server-Anwendungen werden sowohl langlebige Daten (Beispiele: sequentielle Dateien, relationale Tabelle) als auch kurzlebige Daten (Beispiele: Records, Vektoren) bearbeitet. Eine nicht uniforme Behandlung der persistenten und transienten Daten erhöht die Komplexität und den Programmieraufwand in großen Systemen. Sprachen, die Persistenzabstraktion unterstützen, befreien den Anwendungsprogrammierer von Schwierigkeiten beim Übergang zwischen der Bearbeitung der persistenten

und transienten Daten [Matthes, Schmidt 97]. Bei vielen Systemen ist die Persistenz beschränkt, beispielsweise auf wenige Datentypen. Die allgemeinste Form ist die orthogonale Persistenz. Vollständige Persistenzabstraktion [Atkinson, Bunemann 87] erfordert:

- Persistenzunabhängigkeit: Die Form des Programms ist unabhängig von der Langlebigkeit der Daten. Dies bedeutet, daß die Algorithmen, wie z.B. Prozeduren oder Funktionen, auf transiente und persistente Daten als Argumente uniform angewandt werden können.
- Orthogonalität der Datentypen: Alle Datenobjekte benutzen unabhängig von ihrem Typ die volle Bandbreite der Persistenz. Dies bewirkt, daß in einem persistenten Datenmodell die volle Semantik der Programmiersprache Verwendung finden kann.
- Identifikation der Persistenz: Der Mechanismus zur Identifikation persistenter Objekte ist unabhängig vom Typsystem oder von Objektbezeichnern. Dabei wird die referenzielle Integrität für das gesamte System während seiner Lebensdauer gewährleistet.

3.3.6 Massendatenverarbeitung

Unternehmensweite Anwendungen arbeiten in der Regel mit großen Datenmengen, die komplex strukturiert sind. Diese Daten sind auf eine Vielzahl von Datenbanken verteilt und sind dementsprechend unterschiedlich modelliert. Um diese Daten komfortabel handhaben zu können, sind leistungsfähige Sprachelemente zur Datendefinition und Datenmanipulation erforderlich.

Sprachelemente zur Datendefinition

Zur Realisierung von Datenbanken werden Typkonstruktoren für Massendaten (Bulk-Typen) eingesetzt. Die meisten der existierenden Datenbankprogrammiersprachen bieten einen festen Satz von vordefinierten Bulk-Typen zur Realisierung der Datenbanken [Niederee 92]. Dies widerspricht die Erweiterbarkeit und Portabilität der Anwendungen in Client/Server-Umgebungen. Zur Erfüllung dieser Anforderungen sollen Sprachkonzepte zur Modellierung der Massendaten folgende Merkmale besitzen:

- Das definierte Datenmodell soll unabhängig von den Datenbanksystemen einzelner Hersteller sein.
- Es sollte die Möglichkeit bestehen, daß der Benutzer neue Bulk-Typen definiert, die neuen, speziellen Anforderungen angepaßt sind.

Sprachelemente zur Datenmanipulation

Typische Anforderungen datenintensiver Anwendungen wie Datenanfragen und Datenmanipulation werden durch Sprachkonzepte zur Iterationsabstraktion unterstützt. Diese Iterationsabstraktion bietet meistens generische Operationen zur mengenorientierten Selektion, Modifikation und Aggregation der Massendaten. Das Verwalten der Massendaten in einer heterogenen, ständig wachsenden Client/Server-Umgebung erfordert einen hohen Grad an Flexibilität und Abstraktion dieser Operationen:

- Die Operationen sollten erweiterbar sein, um an die Anforderungen neu integrierter Teilsysteme angepaßt werden zu können.

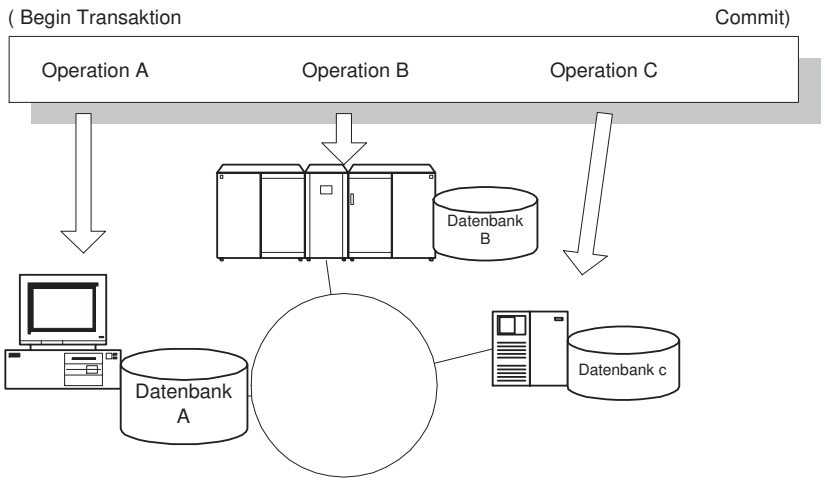

Abbildung 3.5: Eine verteilte Transaktion

- Durch eine Trennung der abstrakten Spezifikation eines mengenorientierten Datenbankzugriffes von seiner Implementierung sollte die erforderliche Abstraktion unterstützt werden [Matthes 93].

- Es sollte möglich sein, verschiedene Sprachelemente zu kombinieren, um komplexe Anfragen einfacher realisieren zu können.

3.3.7 Transaktionen

Das Kennzeichen der Transaktionsverarbeitung ist die konsistente Veränderung eines gemeinsamen Datenbestandes durch mehrere, parallel ablaufende Anwendungen. Formal ist eine Transaktion eine Folge von Verarbeitungsschritten, die die ACID-Bedingung erfüllt (Atomarität, Konsistenz, Isolation, Dauerhaftigkeit).

Diese Eigenschaften machen die Transaktionen zu gewünschten Einheiten im Client-/Server-Computing. Eine Transaktion wird als fundamentale Einheit der Fehlererholung, Konsistenzsicherung und Nebenläufigkeit im Client/Server-System betrachtet. In einer idealen Welt werden alle Client/Server-Programme als Transaktionen entwickelt. Verteilte Transaktionen erstrecken sich über mehrere Knoten und modifizieren die lokalen Ressourcen in jedem Knoten. Verteilte Transaktionen, wie in Abbildung 3.5 dargestellt, benötigen mehr Flexibilität als lokale Transaktionen und führen daher zu folgenden Erweiterungen des lokalen Transaktionskonzeptes:

- Transaktionen sollen in heterogenen Umgebungen unabhängig von speziellen Transaktionsmonitoren ablauffähig sein.

- Es sollen lang andauernde Transaktionen und eine workflow-orientierte Verarbeitung unterstützt werden.

- Ein partielles Zurücksetzen von zusammengesetzten, betriebswirtschaftlichen Transaktionen sollte möglich sein.

Sperrkonzepte

Um ein konsistentes Schreiben und Lesen von Daten durch parallele Benutzer sicherzustellen, werden Sperrkonzepte benötigt. In einer verteilten Umgebung können Datenbankanfragen von mehreren Clients gleichzeitig generiert werden. Der Datenbank-Server soll die Daten vor inkonsistenter Verarbeitung schützen können. Durch Datenbanksysteme werden Sperrmechanismen bereitgestellt, die allerdings nicht für komplexe Objekte gedacht sind. Deshalb sollte die Anwendung die Möglichkeit bieten, komplexe Daten in einer Mehrbenutzerumgebung zu sperren. Unternehmensweite Client/Server-Anwendungen führen zu einem hohen Grad an paralleler Verarbeitung der Daten. Ein vollständig automatisches Sperren der Daten kann diesen Parallelitätsgrad stark einschränken. Deshalb sollte die Entwicklungsumgebung über eine optionale Sperrtechnologie verfügen, die eine höhere Performanz, d.h. weniger gesperrte Daten und einen höheren Grad der Parallelität, ermöglicht.

3.3.8 Programmierung grafischer Benutzerschnittstellen

Eine grafische Benutzerschnittstelle ist ein unabdingbarer Bestandteil jedes Client/Server-Systems für betriebswirtschaftliche Lösungen. Daher hat die GUI-Programmierung bei der Entwicklung unternehmensweiter Client/Server-Anwendungen die gleiche Bedeutung wie die Anwendungsprogrammierung. Die Erstellung von Anwendungsprogrammen mit grafischer Benutzerschnittstelle ist wegen hohen Geschwindigkeits- und Stabilitätsanforderungen, der Mehrprozeßfähigkeit, der Notwendigkeit zur Abhandlung asynchroner Ereignisse und wegen der Portabilitätsanforderungen [Müßig 94] schwieriger als die Erstellung anderer Software. Aus diesen Gründen existieren verschiedene Ansätze für die Erstellung der grafischen Benutzerschnittstelle von Anwendungsprogrammen. Es wird allgemein zwischen *add-on* und *built-in* Ansätzen unterschieden. In einem *add-on* Ansatz werden die Dienste von Bibliotheken (auch kommerzieller Bibliotheken), die die Realisierung von grafischen Benutzeroberflächen für mehrere Systemplattformen ermöglichen, eingesetzt. Die Erweiterung des Programmiermodells um Konzepte zur Darstellung von GUI-Elementen wird als *built-in* Ansatz bezeichnet. Die folgenden Kriterien sollten bei der Auswahl eines Ansatzes berücksichtigt werden:

- Portierbarkeit: Eine Umstellung der GUI-Plattform sollte während der Lebensdauer eines betrieblichen Informationssystems möglich sein. Dies kann durch Fenstersystemunabhängigkeit und Plattformunabhängigkeit der GUI-Programme erreicht werden.

- Verteilbarkeit: Zur Gewährleistung der Verteilbarkeit sind klare Schnittstellen zwischen GUI-Komponente und Anwendungscode einzuhalten. Durch Trennung zwischen Anwendung und Benutzerschnittstelle werden kürzere Entwicklungszeiten und einfacheres Prototying und Verteilung erreicht.

- Standardisierung: Standards für das Aussehen sowie das Verhalten von interaktiven Anwendungen sollen definiert sein.

- Ergonomie: Für die grafische Benutzerschnittstelle werden auch aus ergonomischer Sicht zahlreiche Anforderungen aufgeführt. Von besonderer Bedeutung sind die Richtlinie EU90/270 EG der Europäischen Union, die die Anforderungen an menschengerechte Computerarbeitsplätze festlegt, sowie die ISO-Norm 9241-10, die sieben

Gestaltungskriterien für ergonomische Software definiert und ausführlich kommentiert: Aufgabenangemessenheit, Selbstbeschreibungsfähigkeit, Steuerbarkeit, Erwartungskonformität, Fehlerrobustheit, Individualisierbarkeit und Erlernbarkeit [Buck-Emden, Galimow 95].

3.4 Kriterien für die Werkzeugwahl

Will man die Problemstellung eines Unternehmens softwaretechnisch lösen, so greift man dazu auf Werkzeuge zurück. Die Zeiten, in denen man als die einzigen Werkzeuge Editor, Compiler und Linker zu Verfügung hatte, sind eindeutig vorbei. Heute wird eine Vielzahl von Werkzeugen angeboten, mit denen Softwareentwicklung leichter wird. Leistungsfähige GUI-Builder ermöglichen eine schnelle Gestaltung von aufwendigen grafischen Benutzeroberflächen. Die Anbindung dieser Benutzeroberflächen an das Datenbankmanagementsystem wird mit visuellen Methoden unterstützt. Diese Werkzeuge sind aber leider nur für die schnelle Erstellung von einfachen Lösungen geeignet, die keine aufwendige Anwendungslogik beinhalten.

Unternehmensweite Client/Server-Anwendungen, die Kernabläufe im Unternehmen unterstützen, bestehen aus großen Mengen von Modulen, Transaktionen und Daten, die in verteilten heterogenen Systemumgebungen eingesetzt werden und mit unterschiedlichen Fremdanwendungen kooperieren. Deshalb müssen während des gesamten Softwarelebenzyklus, d. h. Analyse und Design, Implementierung und Wartung dieser Anwendungen, leistungsfähige, portable Werkzeuge vorhanden sein.

In diesem Abschnitt werden die folgenden Werkzeuggruppen zu Unterstützung der Entwicklungsphasen von Client/Server-Software diskutiert:

- Werkzeuge für die Analyse der Geschäftsprozesse
- Werkzeuge zur Implementierung
- Software-Management-Werkzeuge.

3.4.1 Werkzeuge für die Analyse der Geschäftsprozesse

Die Ableitung mehrfach verwendbarer Dienste aus den Geschäftsprozessen des Unternehmens erfolgt in der Analysephase des Client/Server-Projekts. Durch eine werkzeugunterstützte Methodik sollen die Geschäftsprozesse analysiert, Dienste abgeleitet und sie so strukturiert werden, daß eine verteilbare Systemarchitektur entstehen kann.

Dabei sollen folgende Werkzeuge vorhanden sein:

- Werkzeuge zur grafischen Aufbereitung und Verwaltung von Datenflußdiagrammen
- Werkzeuge zur Darstellung der Funktionsablaufdiagramme
- Dokumentationswerkzeuge (Hypertext-Dokumentationen)

3.4.2 Werkzeuge zur Implementierung

Die Werkzeuge der Implementierungsphase eines Softwareprojekts sind im wesentlichen Editor, Compiler und Linker. Die Entwicklung von Client/Server-Anwendungen sollte durch Einsatz folgender Werkzeuge zusätzlich unterstützt werden:

- Entwicklungswerkzeug für grafische Benutzerschnittstellen: Das Aushängeschild einer Client/Server-Anwendung ist seine Benutzeroberfläche. Daher sollte das Oberflächendesign vom Oberflächendesigner und unabhängig von der Anwendung entwickelt werden können. Durch die separate Entwicklung der Oberfläche erreicht man eine Trennung vom Programmcode der Anwendung. Diese ermöglicht nicht nur kurze Entwicklungszeiten sondern auch das einfache Prototyping [Müßig 94].

- Make-Werkzeug: Erweiterbarkeit ist eine wichtige Eigenschaft unternehmensweiter Client/Server-Anwendungen. Dies hat die Änderung der Modulschnittstellen zur Folge. Wenn sich die Schnittstelle eines Moduls ändert, müssen alle Module, die diese Schnittstelle benutzen, neu übersetzt werden. Dabei können unter Umständen Fehler entdeckt werden, weil einzelne Module an die neue Schnittstelle angepaßt werden müssen [Potter, Dr. Cory 93]. Solche Übersetzungen können mit Werkzeugen automatisiert werden. Ein Beispiel dafür ist das Unix Make-Werkzeug. Make benutzt eine Beschreibung von Abhängigkeiten zwischen Modulen und überprüft anhand der Änderungszeitpunkte, welche Module neu übersetzt werden müssen.

- Browser: In besonders großen Programmsystemen, wie es unternehmenensweite Client-/Server-Programme sind, ist es nicht einfach, den Überblick zu bewahren. Eine große Hilfe können sogenannte Browser sein, mit denen die Struktur von Programmen in grafischer Form oder als Text angezeigt werden kann. Solche Browser sind besonders für stark modularisierte Client/Server-Anwendungen nützlich. Sie zeigen die Funktionsaufrufshierarchie oder Vererbungshierarchie grafisch an, geben Auskunft über importierte Modulen usw.

- Debugger: Die Fehleranalyse bei der Entwicklung integrierter Client/Server- Anwendungen kann besonders schwierig werden. Die Fehler können in entfernten Funktionen oder integrierten Anwendungen sein. Daher kann ein leistungsfähiger Debugger eine sehr wichtige Rolle spielen.

- Programmgeneratoren: Für immer wiederkehrende, gleichartige Aufgaben können Programmgeneratoren eingesetzt werden, die leere Programme mit einheitlichen Programmköpfen und Platzhaltern für Kommentare erzeugen. Nützlicher sind Werkzeuge, die aus einer formalen Beschreibung einer Aufgabe ein ganzes Programm automatisch herstellen können, das weiterhin modifizierbar bzw. anpaßbar ist.

- Repository: Überblick über die Zusammenhänge des Gesamtsystems liefern Repositories. Ein Repository ist eine Meta-Datenbank, die Informationen über Unternehmensabläufe, Projektdaten, Designinformationen und Dokumentationen plattformübergreifend für das gesamte Unternehmen enthält. Herauszuheben sind folgende Anforderungen an das Repository [Potter, Dr. Cory 93]:

  - Die physikalische Verteilung der Teilrepositories sollte für den Entwickler transparent bleiben.
  - Die Information über die Topologie des Netzwerks sollte auch im Repository abgelegt werden können.
  - Die Anwendungssprache und die Datenbank sollten bevorzugt auf einem gemeinsamen, integrierten Repository operieren.
  - Benutzerdefinierte Objekte sollten ebenfalls im Repository abgelegt werden können.

  - Schnittstellen zu externen Repositories zwecks Datenaustausch sollten vorgesehen sein.
  - Zur Unterstützung von Teamarbeit sind Mehrbenutzer-Repositories besonders wichtig.
  - Referenzielle Integrität innerhalb vom Repository sollte gewährleistet sein.

- Werkzeuge zur Unterstützung von Workgroup-Funktionalität: Bei der Entwicklung von Client/Server-Anwendungen wirken viele Entwickler an ein und derselben Aufgabe mit. Dies erfordert zusätzliche Werkzeuge zur Unterstützung der Workgroup-Funktionalität.

3.4.3 Software-Management-Werkzeuge

Software-Management-Werkzeuge werden von Entwicklerteams und beim Einsatz von Software in verteilten Umgebungen benötigt. Die wichtigsten Aufgabengebiete sind:

- Versionsverwaltung und Versionsvergleich zwischen Client und Server: Die Versionen der auf dem Client und dem Server laufenden Programme sollten kontrolliert und auf ihre Konsistenz geprüft werden.

- Softwareverteilung im heterogenen Netz: Es sollte die Aufteilung der Anwendung zwischen Client und Server sowie verschiedenen Servern entsprechend des vorgegebenen Architekturmodells unterstützt werden.

- Automatische Lastverteilung: Die Performanz des Client/Server-Systems sollte regelmäßig verfolgt werden. Im Fall einer Überlastung sollte es möglich sein, dynamisch die Anfragen auf nicht belastete bzw. weniger blastete Server umzuleiten.

- Zusammenarbeit mit verschiedenen Ressourceverwaltern: In heterogenen Netzumgebungen existieren eine Reihe von Ressourcenverwaltern. Es sollte eine automatische Zusammenarbeit dieser Ressourceverwaltern miteinander, zur Laufzeit unterstützt werden.

- Zusammenarbeit mit verschiedenen Kommunikationsmechanismen: Es sollte dynamisch die Art und Weise der Kommunikationsmechanismen erkannt und die richtige Schnittstelle eingeschaltet werden. Beispielsweise sollte eine auf ORB basierte Kommunikation erkannt und deren Schnittstelle aufgerufen werden [Orfali et al. 96].

3.5 Korrelation zwischen externen und internen Qualitätsmerkmalen

Die in diesem Kapitel beschriebenen internen Qualitätsmerkmale bieten nun eine Basis für den Systemvergleich. Nach einer Einordnung der internen Qualitätsmerkmale eines Systems und unter Berücksichtigung der besonderen Anforderungen unternehmensweiter Client/Server-Anwendungen, die im Kapitel 4.2 als externe Qualitätsmerkmale formuliert sind, kann eine Produktevaluation durchgeführt werden.

Die tabellarische Darstellung der Korrelation zwischen internen und externen Qualitätsmerkmalen kann die Produktevaluation erleichtern. Dabei kann die relative Gewichtung des individuellen Beitrags eines internen Qualitätsmerkmals zu einem gegebenen ex-

ternen Qualitätsmerkmal als Maßstab bei der Auswahl dieses internen Qualitätsmerkmals als Evaluationskriterium dienen.

Zum Beispiel sollte, wenn die Portabilität bei der Auswahl eines Produktes im Vordergrund steht, in der Korrelationstabelle die Korrelation zwischen Portabilität und den internen Qualitätsmerkmalen betrachtet werden. Aus den Tabellen ist zu sehen, daß korrelieren die internen Qualitätsmerkmale Datenbank-, Betriebssystem- und Netzwerkschnittstelle positiv mit der Portabilität. Daher können diese internen Qualitätsmerkmale als Evaluationskriterien verwendet werden.

Die Gewichtungsfaktoren sind in den Tabellen nicht numerisch dargestellt, weil sie häufig mit einer subjektiven Wertung verbunden sind (z.B. ist es generell nicht möglich zu entscheiden, ob der Unicode-Zeichensatz wichtiger als die TCP/IP-Protokollunterstützung für die Portabilität von Softwaresystemen ist).

In den Tabellen sind daher nur drei Arten der Bewertung unterschieden:

- keine Korrelation (KK),
- korreliert (K) und
- stark korreliert (SK).

Korrelationstabellen sind im Anhang B dargestellt.

Kapitel 4

NATURAL: Eine 4GL-Entwicklungsumgebung

NATURAL, das Anwendungsentwicklungssystem der Software AG, ist ein systemintegriertes Werkzeug zur Entwicklung, Pflege und Dokumentation von datenintensiven Online- und Batch-Anwendungen. Bei der Entwicklung von NATURAL wurde Wert darauf gelegt, daß folgende drei Ziele erreicht werden [Software 90]:

- Portabilität: Unabhängigkeit von der eingesetzten Hard- und Software,
- Integration: Problemlose Einbettung in die vorhandene Umgebung und
- Performanz: Produktivität und Portabilität sollen nicht mit Performanz-Nachteilen verbunden sein.

Mit NATURAL steht eine komplette Umgebung zur Anwendungsentwicklung zur Verfügung. Diese besteht aus einer 4GL zur Programmierung und Wartung von Daten und Programmen, Analyse- und Designwerkzeugen, einem Anwendungsgenerator und Schnittstellen für Kommunikation und Datenaustausch sowie Software-Entwicklung im Team.

In diesem Kapitel werden die internen Qualitätsmerkmale der Entwicklungsumgebung von NATURAL (Version 3.2.1) beschrieben. Der in Kapitel 3 aufgestellte Katalog für interne Qualitätsmerkmale einer Entwicklungsumgebung wird als Rahmen für die Struktur dieser Beschreibung verwendet. Im ersten Teil wird die Systemarchitektur von NATURAL dargestellt. Anschließend wird das Programmiermodell der 4GL-NATURAL beschrieben. Im letzten Teil werden die Werkzeuge zur Erstellung und Pflege von verteilten Anwendungen vorgestellt.

Diese Beschreibung basiert zum großen Teil auf Dokumenten und Publikationen der Software AG und persönlicher Programmiererfahrung der Autorin mit NATURAL und NATURAL *LightStorm*.

4.1 Die NATURAL-Systemarchitektur

Durch seine Architektur will NATURAL als Integrator zwischen den im Unternehmen vorhandenen Daten und dem Endbenutzer fungieren. Ein entscheidendes Kriterium für dieses Konzept ist die Offenheit der Architektur. Realisiert wird das Konzept durch die Definition von Schnittstellen, die die Kommunikation mit der Außenwelt ermöglichen. In der Abbildung 4.1 sind die Komponenten der offenen NATURAL-Architektur dargestellt. Im folgenden werden diese Komponenten näher erläutert.

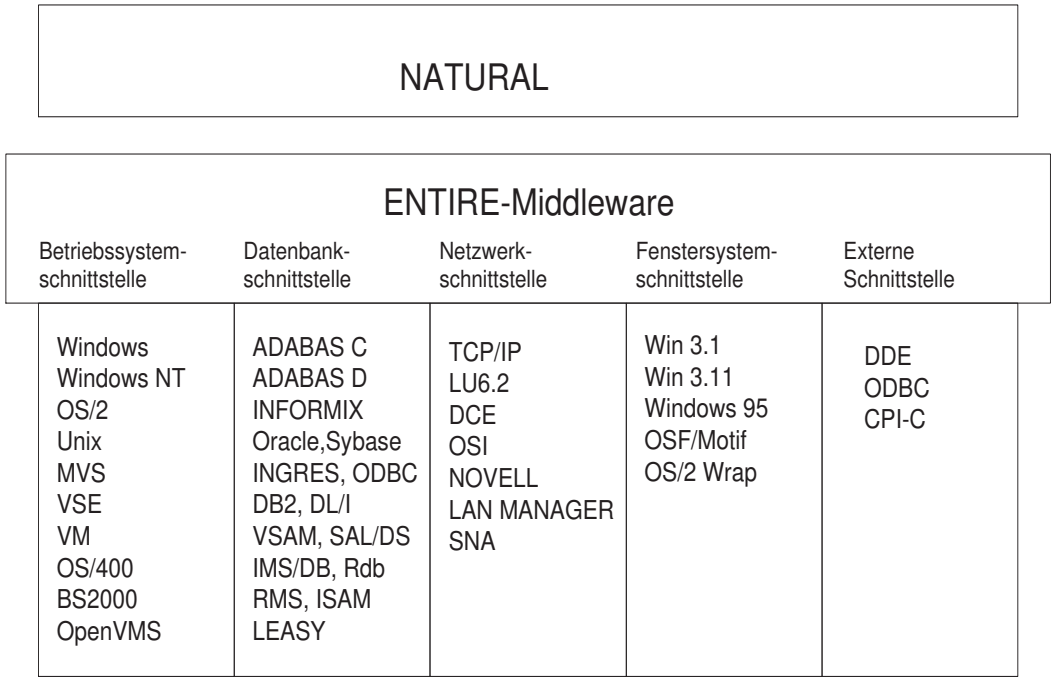

Abbildung 4.1: Die offene Architektur von NATURAL

4.1.1 Betriebssystemschnittstelle

NATURAL-Programme lassen sich ohne Änderung bzw. Generierungsaufwand in unterschiedlichen Betriebssystemen verschiedener Hersteller ausführen. NATURAL-Anwendungen mit grafischer Benutzeroberfläche sind auf Windows, OS/2 und OSF/Motif portierbar. Auf dem Mainframe entwickelte Anwendungen und Programmteile können ohne weiteres direkt auf den PC portiert werden.

4.1.2 Netzwerkschnittstelle

Die Kommunikationsschnittstelle, die durch ENTIRE NET-WORK realisiert wird, definiert die Verteilungs-und Kommunikationsdienste für unterschiedliche Anforderungen, wie verteilte Datenbanken und Programm-zu-Programm-Kommunikation. ENTIRE NET-WORK bietet eine einheitliche Schnittstelle für heterogene Plattformen und unterschiedliche Netzwerkumgebungen. Die Abbildung 4.2 stellt den Einsatz von ENTIRE NET-WORK in einer heterogenen Client/Server-Umgebung dar [Software 96b].

Unterstützt werden die herstellereigenen Protokolle wie IBMs SNA, Digitals DECnet, SIEMENS DCAM, aber auch OSI, TCP/IP, LU6.2, AppleTalk, sowie Microsoft LAN Manager. NATURAL-Anwendungen arbeiten auf der logischen Ebene völlig unverändert, unabhängig davon, welches System physisch eingesetzt wird.

4.1.3 Datenbanksystemschnittstelle

NATURAL ermöglicht NATURAL-Anwendungen den Zugriff auf unterschiedlichste Datenbanksysteme wie z.B. ADABAS, DB2, DL/I etc. Die NATURAL Datenbankschnittstelle generiert die entsprechende Zugriffssyntax transparent für den Entwickler. Dadurch

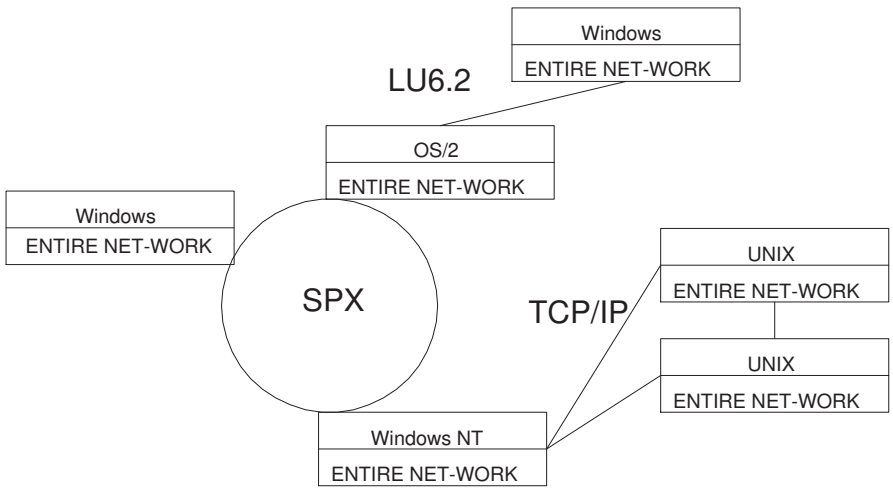

Abbildung 4.2: Netzwerk-Integration in NATURAL

ist die Portabilität über heterogene Systeme erreicht. Der Zugriff auf Datenbanken kann wie folgt erreicht werden:

- über die ADALINK-Schnittstelle [1] auf eine lokale ADABAS-Datenbank,
- über die ODBC-Schnittstelle auf andere lokale Datenbanken,
- über ENTIRE NET-WORK und ADALINK auf alle entfernten ADABAS-Datenbanken,
- über ENTIRE-ACCESS und ENTIRE NET-WORK auf entfernte SQL-basierte Datenbanken.

4.1.4 Schnittstellen zu externen Anwendungen

NATURAL erzielt Interoperabilität durch Unterstützung etablierter Standards. Dazu gehören:

- Schnittstelle zu Desktop-Anwendungen: Für die Anbindung von Windows-Anwendungen unterstützt NATURAL DDE (Dynamic Data Exchange) und ODBC für den Bereich des Datenzugriffs.
- CPI-C-Schnittstelle (Common Programming Interface Communication) für die Kopplung mit SAA-Application.

4.1.5 Schnittstelle zu anderen Sprachen

Die CALL-Schnittstelle in NATURAL ermöglicht den lokalen Aufruf des in einer anderen Programmiersprache geschriebenen Programms von einem NATURAL-Programm. Das aufgerufene Programm kann in einer beliebigen anderen Programmiersprache, die eine Standard-CALL-Schnittstelle unterstützt, geschrieben sein. Zur Zeit unterstützen die Programmiersprachen COBOL, PL/I und C diese Schnittstelle. Mit der CALL-Anweisung

[1] ADALINK ist eine *Application Programming Interface*.

kann von einem NATURAL-Programm aus ein externes Programm aufgerufen werden, wobei anschließend die Verarbeitung des NATURAL-Programms mit der nächsten Anweisung nach der CALL-Anweisung fortgesetzt wird. Der Name des aufgerufenen Programms kann entweder als Konstante oder als Variable angegeben werden. Die CALL-Anweisung kann bis zu 40 Parameter erhalten. Die Parameter können beliebige atomare Datentypen besitzen, die in NATURAL vordefiniert sind. Es ist auch möglich, Konstanten, Arrays, Strukturen und NATURAL-Systemvariablen als Parameter zu definieren. Der Returncode kann mit Hilfe der Systemfunktion RET(programmname) erhalten werden. Der Rückgabewert kann nur eine 4-Byte Integer-Zahl sein. Das aufgerufene Programm kann entweder direkt an den NATURAL-Nukleus gelinkt werden, oder es kann dynamisch geladen werden [Software 93].

4.1.6 Fenstersystemschnittstelle

Diese Schnittstelle sorgt für die einheitliche *Look and Feel* der grafischen Oberflächen gegenüber dem Endbenutzer und ermöglicht die Darstellung grafischer Benutzeroberfläche auf Windows, OS/2 und OSF/Motif.

4.2 Das NATURAL-Programmiermodell

Die 4GL-NATURAL ist eine höhere Programmiersprache mit prozeduralen und nichtprozeduralen Sprachelementen. Sie ist Teil eines Gesamtsystems, das in eine Betriebssystem- und Monitorumgebung eingebunden ist, und Datenbankzugriffe ebenso umfaßt wie Betriebssystemzugriffe. Das wichtigste Merkale von NATURAL ist die über verschiedene heterogene Umgebungen einheitliche Syntax und Semantik. Die Sprache steht auf den wichtigsten Plattformen im kommerziellen Umfeld zur Verfügung. Dadurch ist es möglich, Client/Server-Anwendungen zu entwickeln, die nicht nur portiert, sondern auch verteilt werden können. In diesem Abschnitt wird auf die Programmierung in der NATURAL-Sprache eingegangen.

4.2.1 Datenstrukturierungskonzepte

NATURAL bietet eine große Anzahl von vordefinierten atomaren Datentypen und dazugehörigen vordefinierten Operationen. Der Benutzer kann ausschließlich Variablen dieser Typen definieren. Die Definition neuer Datentypen bzw. abstrakter Datentypen ist in der Sprache nicht vorgesehen. Das Typsystem ist monomorph, und es wird keine Generalisierung zugelassen. Im folgenden werden die vordefinierten atomaren und die Strukturierungsmöglichkeiten für benutzerdefinierte Variablen näher erläutert.

Vordefinierte atomare Datentypen

NATURAL verfügt über einen Satz vordefinierter Datentpen zur Darstellung von Zahlen, Text, Zeit, Datum, logischen Wahrheitswerten usw. Die Darstellung von Zahlen erfolgt in zahlreichen Formen, wie Floating, Integer, Numeric und packed Numeric. Diese Datentypen decken betriebswirtschaftliche und wissenschaftliche Berechnungen ab. NATURAL bietet zur Datums- und Zeitrechnung spezielle Typen und eingebaute Operationen. Das Datumsformat kann europäisch, deutsch, international oder amerikanisch eingestellt werden. NATURAL verfügt über zahlreiche Operationen zur Verarbeitung von Strings, z.B. MASK, SCAN, EXAMINE zur Teilstringsuche und Teilstring-Analyse oder COMPRESS

zum Übertragen zweier oder mehrerer String-Felder in ein String-Feld. Die Automatische Konvertierung unterstützt die Übertragung der Daten in ein Feld mit einem anderen Format. Numerische Variablen anderer Formate können nicht in Datums- und Zeitvariablen übertragen werden. Mit einer IS-Operation kann geprüft werden, ob der Inhalt eines Feldes in ein bestimmtes anderes Datenformat übertragbar ist [Software 93].

Typkonstruktoren

NATURAL verfügt über keine Typkonstruktoren. Es sind lediglich Strukturierungsmöglichkeiten für die Variablen vorhanden. Durch diesen Mangel gehen die Typinformationen verloren. In NATURAL können allerdings die Definitionen von Variablen in Datenbereichen als Copycode wiederverwendet werden (s. Abschnitt 4.2.3). Dies hat höheren Wartungsaufwand zur Folge und erschwert die Erweiterbarkeit der Software. In folgenden werden Strukturen und Arrays, die die Strukturierung von Variablen ermöglichen, näher erläutert.

Strukturen

In NATURAL können Variablen unter einem Strukturnamen zusammengefaßt und unter Angabe des Strukturnamens angesprochen werden. Die Charakteristika der Strukturen in NATURAL können wie folgt zusammengefaßt werden:

- Die Komponenten einer Struktur können Variablen beliebiger Datentypen, Konstanten, Tabellen und Strukturen sein. Die Definition einer Funktionsabstraktion als Komponente einer Struktur ist nicht zugelassen.
- Ein selektiver lesender und schreibender Zugriff auf die Komponenten ist sowohl über Punktnotation als auch direkt über Variablennamen möglich. Da derselbe Feldname in mehreren Strukturen oder in Datenbanksichten verwendet werden kann, kann die direkte Identifikation eines Feldes problematisch werden, falls für eine Benutzervariable und ein Datenbankfeld derselber Namen verwendet wird.
- Korrespondierende Komponenten zweier Strukturen können mit einem Befehl zugewiesen werden. Die Spezifikation der Felder geschieht über ihre Namen oder ihre Position innerhalb der Struktur.
- Ein Feld einer Struktur kann durch die REDEFINE-Option redefiniert werden. Die REDEFINE-Option redefiniert die Byte-Positionen eines Feldes von links nach rechts, unabhängig vom Format. Die Byte-Positionen des ursprünglichen Feldes und des neu definierten Feldes müssen einander entsprechen. Redefinition dient dazu, den Typ eines Feldes zu ändern oder ein einzelnes Feld in mehrere Teile aufzuteilen. Eine Struktur kann ebenfalls redefiniert werden.

Array

Arrays in NATURAL sind mit dem aus Pascal bekannten Array-Konstruktor vergleichbar. Arrays sind mehrdimensionale Tabellen, die unter einem gemeinsamen Namen definiert werden. Arrays können aus einzelnen Datenelementen mit bis zu drei Dimensionen oder aus Datenstrukturen bestehen. Eine Tabelle kann wie ein einfaches Datenfeld definiert werden. Es muß jedoch zusätzlich ein Index oder Indexbereich zur Festlegung der Kardinalität und der Anzahl der Dimensionen angegeben werden. Die Tabellen können maximal 32K groß sein. Zur Verarbeitung von Arrays gelten folgende Regeln:

- Alle Skalaoperationen können auf Array-Elemente angewandt werden.
- In arithmetischen Operationen (COMPUTE, ADD, MULTIPLY) können Array-Bereiche mit gleichen Dimensionen als Argument verwendet werden. Die Operation wird Element für Element ausgeführt.
- Es ist möglich, ein einzelnes Element einem Array-Bereich zuzuweisen. Jedes Element des Bereiches wird dann mit dem Wert der einzelnen Ausprägung gefüllt.
- Eine Zuweisung bzw. ein Vergleich zwischen zwei Arrays mit unterschiedlich vielen Dimensionen ist möglich. Die fehlende Dimension in dem Array mit weniger Dimensionen wird automatisch mit einem Element versehen.

4.2.2 Strukturierungstechniken für die Programmierung im Kleinen

Neben den deskriptiven Sprachelementen für Datenbankzugriffe und Datenbankänderungen bietet NATURAL auch Kontrollstrukturen zur Realisierung imperativer Algorithmen. Dies erlaubt die Entwicklung der Anwendungslogik und Steuerungskomponente von Client/Server-Anwendungen. Die Beschreibung dieser Kontrollstrukturen ist der Gegenstand dieses Abschnittes.

Blockstrukturierte Kontrollstrukturen

Für verschiedengeartete Entscheidungen (Verzweigungen, Fallunterscheidungen) bietet NATURAL die Auswahl zwischen IF-Anweisung oder DECIDE-Konstrukten. Mit DECIDE-Konstrukten können auch komplexe Logikabfragen relativ transparent für den Programmleser gestaltet werden. NATURAL kennt zwei Arten von Schleifen: Datenbankschleifen und Nichtdatenbankschleifen. Nichtdatenbankschleifen (REPEAT, FOR, CALL LOOP) fassen eine Folge von NATURAL-Anweisungen zwecks Iteration zusammen. Zu einer Zeit können mehrere Schleifen aktiv sein. Eine Schleife kann zwischen anderen eingeschlossen oder in anderen aktiven schleifen enthalten sein. Schleifen können mit Erfüllung einer logischen Bedingung enden. Ihre Durchführung kann aber auch mit ESCAPE oder TERMINATE abgebrochen werden.

Prozedurale Abstraktion

Sourcecodeteile, die innerhalb eines Dialogs oder Programms mehrmals notwendig sind, können in einer Inline-Subroutine einmal codiert und dann mehrfach referenziert werden. Inline-Subroutine bieten die einfachste Form der prozeduralen Abstraktion. Sie können nicht parametrisiert werden und verfügen über keine lokale Daten. Eine Inline-Subroutine kann auf alle Daten des aufrufenden Programms zugreifen. Eine Subroutine kann von mehreren Stellen im Programm aufgerufen werden. Eine Inline-Subroutine kann auch innerhalb einer anderen Inline-Subroutine aufgerufen werden. Sie kann auch sich selbst aufrufen. Die Inline-Subroutine werden mit der END-SUBROUTINE-Anweisung oder zuvor mit ESCAPE ROUTINE beendet. Danach wird die nächste Anweisung im Hauptprogramm nach dem Subroutine-Aufruf abgearbeitet.

Ausnahmebehandlung

Zur Behandlung eines zur Laufzeit aufgetretenen Fehlers gibt es in NATURAL mehreren, sich ergänzende Möglichkeiten mit unterschiedlichen Auswirkungen auf den weiteren

Programmlauf. Fehler können wie folgt abgefangen werden [Scheu 95]:

- mit ON ERROR-Block-Anweisung
- mit einer ausgewählten (benutzerdefinierten) Fehlertransaktion (Standardfehlerroutine). Die Systemvariable ERROR-TA enthält den Namen der für die Anwendung aktiven (definierten) Fehlertransaktion.

Ein Fehler wird wie folgt behandelt:

- Rekursiv (von innen nach außen) werden alle Programme und Unterprogramme nach einem ON ERROR-Block gesucht.
- Existiert ein ON ERROR-Block, wird die darin beschriebene Codesequenz durchgeführt.
- Existiert kein ON ERROR-Block, werden alle Programme und Unterprogramme von innen nach außen geschlossen.
- Falls die Systemvariable ERROR-TA mit dem Namen einer Fehlertransaktion (Fehlerroutine) gefüllt ist, wird diese aufgerufen.
- Ist die Systemvariable ERROR-TA nicht belegt, wird die Ausführung der Anwendung mit einer NATURAL-Fehlermeldung unterbrochen.

Die ON-ERROR-Anweisung ist non-prozedural. Das heißt, seine Ausführung hängt von einem Ereignis ab, nicht davon, wo im Programm es steht. Jedes NATURAL-Objekt darf maximal eine ON-ERROR-Anweisung enthalten. Standardmäßig werden Fehlermeldungen über die NATURAL-Bibliothek SYSERR gepflegt.

Die Behandlung der Ausnahmen, die in einer entfernten Funktion auf dem Server ausgelöst werden, wird auch auf diese zwei Möglichkeiten abgebildet. Abhängig davon, wie die Parameter für die Fehelbehandlung eingesetzt ist, wird ein Fehler im entfernten Subprogramm wie folgt behandelt:

- Wenn dieser Parameter mit N belegt ist, sendet NATURAL RPC die Fehlernummer, d.h. den Inhalt vom Systemvariable ERROR-NR, zurück. Auf der Client-Seite wird nun eine lokale Ausnahmesituation simuliert.
- Wenn dieser Parameter mit Y belegt ist, sendet NATURAL RPC die Fehlerinformationen und einen Fehler-Text zurück. Auf der Client-Seite wird dann die aktuelle Fehlertransaktion, deren Namen im Systemvariable ERROR-TA erhalten ist, aufgerufen.

4.2.3 Strukturierungskonzepte für die Programmierung im Großen

In NATURAL sind Modularisierungskonzepte zur Strukturierung großer Anwendungen vorgesehen. Eine typische NATURAL-Anwendung besteht nicht aus einem einzigen großen Programm, sondern ist in mehrere Module aufgeteilt. Die mehrfache Verwendung von Code wird nur eingeschränkt und durch Herausfaktorisieren eines Anweisungsblocks, der in identischer Form in mehreren Programmteilen erscheinen soll, unterstützt. In diesem Abschnitt werden die Konzepte zur Modularisierung und Wiederwervendung von Code in NATURAL vorgestellt.

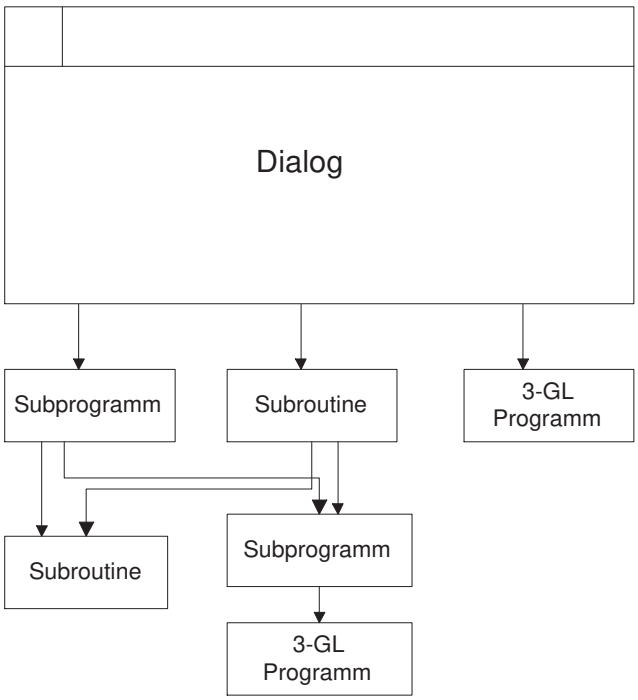

Abbildung 4.3: Unterprogrammtechniken in NATURAL

Modularisierungskonzepte

NATURAL bietet zur Modularisierung verschiedene Programmobjekttypen an. Häufig benötigte Verarbeitungsfolgen wie Ausgabesteuerung, Berechnungen, Hilfeaufrufe usw. können in Module ausgelagert werden. NATURAL kennt verschiedene Formen von Unterprogrammen, die sich in der Form der Parameterübergabe, des Aufrufs und der Strukturierungsmöglichkeiten unterscheiden. In Abbildung 4.3 sind die NATURAL-Objekte zur Modularisierung zusammengefaßt.

Ein Hauptprogramm, das ausgeführt wird, kann andere Programme, Subprogramme, Subroutinen aufrufen. Diese Objekte können ihrerseits wiederum andere Objekte aufrufen. Interne Subroutinen können benutzt werden, um den Wartungsaufwand zu reduzieren, wenn der gleiche Code in mehreren Stellen im Programm benötigt wird. Diese werden innerhalb des Programms in einem Block definiert und können dann mehrmals aufgerufen werden. Wenn eine interne Subroutine so groß ist, daß sie die Lesbarkeit des Objekts, in dem sie enthalten ist, beeinflußt, sollten externe Subroutinen definiert werden. Externe Subroutinen werden als separates Objekt außerhalb des Objektes, das sie aufruft, definiert. Ansonsten sind aus Gründen der Datenkapselungen, Test- und Wartungsvereinfachung Subprogramme vorzuziehen. Darüber hinaus sind Subprogramme entfernt aufrufbar [Scheu 95].

Subprogramm

NATURAL-Objekte vom Type SUBPROGRAM enthalten eine gekapselte Funktionalität. Ein Subprogramm kann nicht selbständig ausgeführt werden. Es muß von einem anderen Objekt aufgerufen werden. Sie werden über die Anweisung CALLNAT von DIALOG-,

PROGRAM-, SUBROUTINE- und SUBPROGRAM-Objekten aufgerufen. Der Datentransfer erfolgt ausschließlich über Parameter. Die übergebenen Parameter müssen im SUBPROGRAM mit DEFINED DATA PARAMETER definiert sein. Die Namen der Parameterfelder im Haupt- und Unterprogramm können unterschiedlich sein. Die Felder müssen jedoch entsprechend ihrer Reihenfolge in Länge und Format übereinstimmen. Parameterdatenelemente dürfen keine Konstanten oder Viewdefinitionen (s. Abschnitt 4.2.5) enthalten.

Das Subprogramm hat keinen Zugriff auf die globalen Daten des Hauptprogramms. Bei Beendigung des SUBPROGRAMS erfolgt ein automatischer Rücksprung in das aufrufende Programm-Objekt. Ein Rücksprung kann über die Anweisung ESCAPE ROUTINE erzwungen werden. Der Aufruf von Subprogrammen kann lokal oder entfernt erfolgen.

Die Zugriffsroutinen in einer datenintensiven Anwendung werden meistens als Subprogramme realisiert. Dabei werden alle Datenzugriffe, die aus Dialog-, Maskensteuerungsprogrammen, Prüfroutinen oder Berechnungsroutinen notwendig sind, in Subprogramms gekapselt. Mit ein CALLNAT aufgerufenes Subprogramm und das rufende Objekt sind voneinander unabhängig. Speziell in Client/Serverumgebungen ist dies ein wichtiges Gestaltungselement. Die Unterprogramme werden in Bibliotheken verwaltet. Normalerweise werden die Unterprogramme, die einheitliche Systemservices oder Anwendungen realisieren, in einer Bibliothek abgelegt.

Externe Subroutine

NATURAL-Objekte vom Typ SUBROUTINE enthalten häufig benutzte Codestrecken oder Routinen. Sie werden über die Anweisung PERFORM vom Dialog- Programm-, Subroutine-, Subprogramm-Objekt aufgerufen. Datentransfer erfolgt über die globalen Daten und/oder durch Parameterübergabe. Damit ist ein Zugriff sowohl auf globale Daten als auch über die Parameterliste auf lokale Daten des rufenden Programms möglich. Eine externe Subroutine kann eigenen lokale Daten besitzen, die nur innerhalb der Subroutine sichtbar sind. Eine externe Subroutine kann keine eigene globale Daten haben. Bei Beendigung oder Verlassen der SUBROUTINE erfolgt ein automatischer Rücksprung in das aufrufende Programm-Objekt.

Wiederverwendung von Code

Neben den oben erwähnten Modularisierungkonzepten, die die Auslagerung häufig wiederholter Verarbeitungsfolgen aus einem Programm und ihre wiederholte Verwendung ermöglichen, kann in NATURAL durch Definition von Kopierstrecken (Copycodes) und Datenbereichen Code gespart werden.

Copycode

Sind Sourcecodeteile verschiedener Objekte identisch, so sollten diese in einem Objekt vom Typ COPYCODE einmal erstellt und dann referenziert werden. Der Sourcecode wird vom Modul getrennt aufbewahrt und in das Objekt mit eingebunden. Dies erhöht die Lesbarkeit des Programms und sichert gleichzeitig die Wiederverwendbarkeit von Sourcecodeteilen. Ein klassischer Anwendungsfall ist die Definition von Funktionstasten. Der Copycode wird bei der Kompilierung statisch in den Objektcode des Objektes, das das INCLUDE-Statement enthält, eingefügt. Wird der Sourcecode des Copycodes verändert, müssen alle Objekte, die den Copycode referenzieren, erneut kompiliert werden.

Datenbereiche

NATURAL stellt für die Definition der Daten neben der Möglichkeit, diese direkt im Modul vorzunehmen, nach Datentypen getrennte Datenbereiche zur Verfügung. Diese Datenbereiche können als eigenständige NATURAL-Objekte erstellt und mehreren Programmen bzw. Unterprogrammen statisch zugeordnet werden. Dies sind:

- Globaler Datenbereich (GDA): Der globale Datenbereich enthält die Datenelemente, die von mehr als einem NATURAL-Objekt referenziert werden können. Pro Programm kann nur ein globaler Datenbereich referenziert werden, dieser muß sich in der gleichen Bibliothek befinden wie das Programm selbst.

- Parameterbereich (PDA): Ein Parameterbereich enthält die in einem Subprogramm oder einer externen Subroutine als Parameter verwendeten Datenelemente. Parameterdatenbereiche können im Programm auch als lokale Datenbereiche verwendet werden. Damit kann ein einmal definierter Parameterbereich im Hauptprogramm als Parameterbereich und im aufgerufenden Subprogramm als lokaler Datenbereich referenziert werden.

- Lokaler Datenbereich (LDA): Die Datenelemente eines lokalen Datenbereichs gelten nur in dem sie referenzierenden Objekt.

4.2.4 Verteilungsabstraktion

NATURAL ermöglicht die kooperative Verarbeitung durch Verteilung von Subprogrammen über unterschiedliche Rechnerplattformen. Interoperabilität zwischen den verteilten Subprogrammen wird durch die Technik des *Remote Procedure Call* unterstützt. Der Zugriff auf entfernte Datenbanken wird vollständig durch die ENTIRE Middleware geleistet. In NATURAL sind nur Daten mobil und über das Netz austauschbar. Im Folgenden werden die Techniken in NATURAL vorgestellt, die die Verteilung von Datenbanken und Subprogrammen ermöglichen.

Entfernter Datenbankzugriff

ENTIRE NET-WORK vermittelt den entfernten Datenzugriff (RDA) auf ADABAS- Datenbanken für NATURAL-Anwendungen. Datenbankzugriffe werden transparent für die Anwendung abgewickelt. Eine ADABAS-Datenbankabfrage wird immer an die Datenbankschnittstelle ADALINK weitergeleitet, die die Abfrage weiter an ADABAS-Datenbank vermittelt. Wenn die Datenbank nicht lokal auf dem Rechner registriert ist, wird die Abfrage über NET- WORK ACCESS-Schnittstelle an ENTIRE NET-WORK weitergegeben, der die entfernte Datenbank lokalisiert und die Anfrage über das Netz an die Ziel-Datenbank sendet. Abbildung 4.4 stellt diesen Vorgang grafisch dar.

Damit ermöglicht die ADALINK-Datenbankschnittstelle einen transparenten Datenbankzugriff auf eine entfernte Datenbank. ENTIRE NET-WORK unterstützt keine verteilten Transaktionen. Jede Datenbankabfrage sollte eine END OF TRANSAKTION-Anweisung beinhalten. Eine einzelne Transaktion kann sich nicht auf mehrere Datenbanken beziehen.

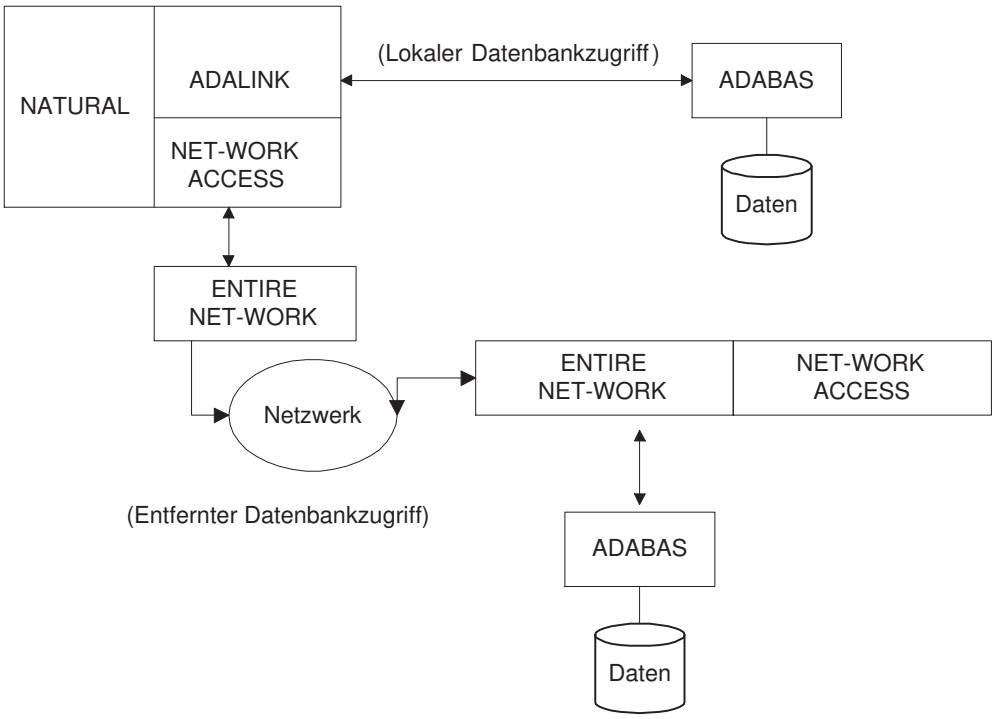

Abbildung 4.4: Entfernter Datenbankzugriff in NATURAL

Entfernter Funktionsaufruf

NATURAL-RPC ist ein integrierter Teil von NATURAL und ist auf Mainframe, Unix, OS/2, Windows und Windows NT vorhanden. Windows kann allerdings nur als Client fungieren. Die NATURAL- RPC ermöglicht die synchrone Ausführung eines NATURAL-Subprogramms auf einem entfernten Server. Der Aufruf von Subprogrammen kann in einer Client/Server-Umgebung lokal oder entfernt erfolgen. Entfernte Aufrufe werden, wie es in Abbildung 4.5 dargestellt ist, durch die ENTIRE-Infrastruktur ermöglicht [Software 96a].

Ein NATURAL-Programm ruft ein Subprogramm zum Erarbeiten eines Ergebnisses auf. Am CALLNAT-Aufruf des Service-Subprogramms ist nicht erkennbar, ob die Leistung lokal oder entfernt erbracht wird. Im Fall eines entfernten Programmaufrufs ist ein Stellvertreter (Stub) auf der Client-Plattform für die Kommunikation mit dem entfernten Subprogramm zuständig. Intern unterscheiden sich die Aufrufe einer lokalen und einer entfernten Prozedur. Ein lokaler NATURAL Prozeduraufruf erfolgt über Referenzparameter (*call by reference*), ein entfernter Prozeduraufruf über Wertparameter (*call by value*). Um Fehler bei der Umstellung von lokaler auf entfernte Verarbeitung zu vermeiden und aus Gründen der Wartbarkeit und leichteren Portierbarkeit der Unterprogramme, sollte eine veränderbare Variable nicht mehrfach an das Subprogramm übergeben werden. Darüber hinaus unterscheidet sich eine entfernte Ausführung eines Subprogramms mittels NATURAL-RPC in folgenden Punkten von einer lokalen Ausführung:

- Eine auf Client-Seite geöffnete Transaktion kann nicht auf Server-Seite geschlossen werden und umgekehrt.

- Das Report-Handling, Work-File Processing und NATURAL Stack können ebenfalls

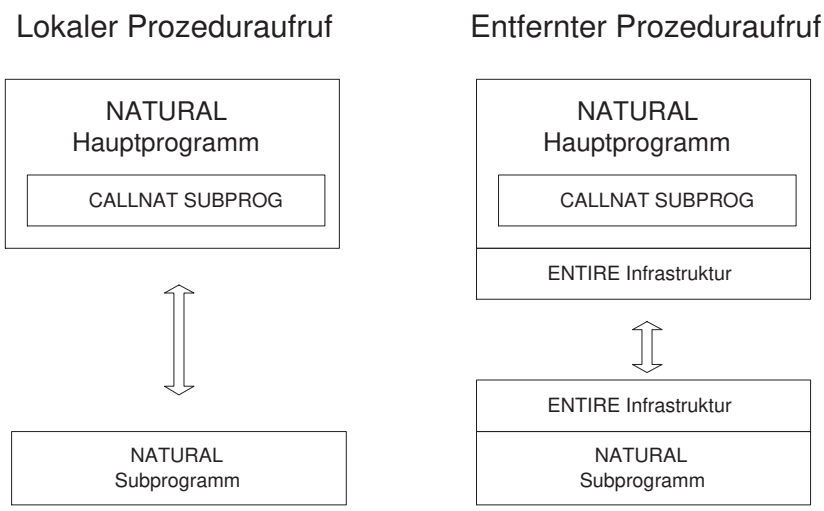

Abbildung 4.5: NATURAL-RPC

nicht auf Server-Seite fortgesetzt werden und umgekehrt.

- Keine Systemvariable kann von der Client-Seite auf die Server-Seite transferiert werden.
- Die Parameter bleiben bei einem fehlerhaften Programmaufruf unverändert.

4.2.5 Massendatenverarbeitung

Die Programmiersprache NATURAL verfügt über eingebettete und eingebaute Datenbankfunktionalität. Die Konzepte zur Unterstützung datenintensiver Anwendungen sind als NATURAL-Data Manipulation Language (DML) in die Sprache fest eingebaut. NATURAL-DML ermöglicht den Zugriff auf unterschiedlichste Datenbanksysteme. Für den Zugriff auf SQL-basierte Datenbanksysteme können auch SQL-Befehle benutzt werden, die ohne Änderung statisch oder dynamisch ausführbar sind. In diesem Abschnitt werden die NATURAL-Sprachelemente zur Definition und zur Manipulation von persistenten Massendaten vorgestellt.

Sprachelemente zur Datendefinition

Ein NATURAL-Programm referenziert nicht die physischen Datenstrukturen, sondern logische Datenstrukturen. Die logischen Datenstrukturen heißen Datendefinitionsmodul (DDM). Die DDM sind die NATURAL-Sicht auf die Daten und stellen die Verbindung zwischen Programm und externen Ressourcen dar. Das heißt, bevor ein in NATURAL geschriebenes Programm auf eine Datei zugreifen kann, muß eine NATURAL-Sicht existieren. Diese Sicht kann alle oder bestimmte Felder der physischen Datei umfassen. Im Programm wird dann die Sicht der Anwendungsfunktion durch den NATURAL Data View beschrieben, der den DDM-Namen referenziert. An dieser Stelle kann eine weitere Beschränkung auf bestimmte Felder einer NATURAL-Sicht vorgenommen werden. Dies ermöglicht die Unabhängigkeit einer NATURAL-Anwendung vom Datenhaltungssystem. Darüber hinaus kann eine NATURAL-Anwendung nur die Felder bearbeiten, die in den

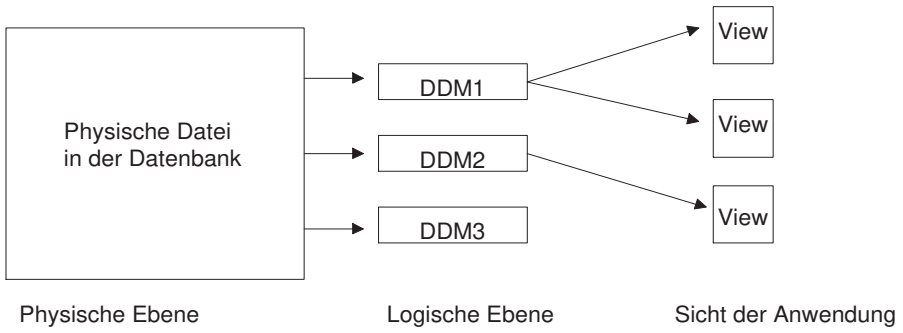

Abbildung 4.6: NATURAL-DDM-Konzept

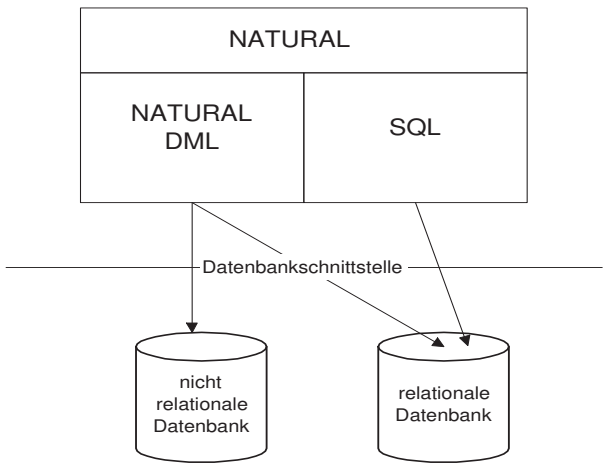

Abbildung 4.7: Sprachelemente zu Datenmanipulation in NATURAL

zur Verfügung stehenden DDMs definiert sind. So gewährleistet das DDM auch den Datenschutz. Das NATURAL-DDM-Konzept ist in Abbildung 4.6 dargestellt [Software 93].

Sprachelemente zur Datenmanipulation in NATURAL

NATURAL erlaubt den mengenorientierten Zugriff auf alle unterstützten Datenbanksysteme. Um aus einem NATURAL-Programm heraus auf Datenbanken zugreifen zu können, existieren grundsätzlich zwei Möglichkeiten:

- NATURAL-Data Manipulation Language (DML)
- NATURAL-SQL Statements

In dem Folgenden werden diese zwei Ansätze zur Datenmanipulation in NATURAL erläutert:

NATURAL-DML

NATURAL besitzt eine eigene Syntax für Datenbankzugriffe, die NATURAL- DML. Mit ihr sind Zugriffe auf verschiedene Datenbanksysteme, relationale und nicht-relationale, möglich. Die NATURAL-DML-Anweisungen werden in die jeweilige Zielsprache des Datenbanksystems umgesetzt, u.a. auch SQL beim Zugriff auf relationale DBMS. Die NATURAL-DML ist für den Zugriff auf unterschiedliche Datenbanksysteme wie ADABAS, DL/I, IMS/DB oder VSAM identisch. Diese Funktionalität ermöglicht, daß aus einer Anwendung gleichzeitig unter Verwendung einer einheitlichen Syntax auf alte IMS/DB-Daten und neue DB2-Daten zugegriffen werden kann. Dadurch ist die Portabilität über heterogene Systeme gewährleistet. Darüber hinaus erleichtert die Datenbankunabhängigkeit der Anwendungen die Erweiterung der Datenbanksysteme ohne Änderung der Anwendungen sowie die Ersetzung von Datenbanksystemen durch modernere Datenbanksysteme. Die NATURAL-DML eignet sich besonders für den Zugriff auf ADABAS- Datenbanken, weil sie speziell auf die Funktionalität dieser Datenbanken abgestimmt ist.

Visible SQL

Auf SQL-basierende Datenbanken ermöglichen NATURAL den Zugriff mit Standard-SQL-Syntax. Auch hier steht dem Programmierer die mengenorientierte Verarbeitung zur Verfügung. Darüber hinaus wird der gesamte Mechanismus zur Steuerung der Cursorlogik von NATURAL automatisch im Hintergrund erzeugt. Der Programmierer codiert nur die reine, sichtbare SQL-Anweisung. Man unterscheidet zwei Syntax-Gruppen:

- Common Set: entspricht der Syntaxdefinition von Standard-SQL. Diese verstehen alle von NATURAL unterstützten SQL-fähigen Datenbanksysteme.

- Extended SQL: Erweiterungen zur Unterstützung spezieller Funktionen für verschiedene, von NATURAL unterstützte Datenbanksysteme.

4.2.6 Transaktionen

NATURAL führt Veränderungen auf der Datenbank auf der Grundlage von Transaktionen aus, d.h. alle Veränderungszugriffe werden in logische Transaktionen gegliedert. Eine Transaktion ist die kleinste vom Programmierer definierte Verarbeitungseinheit, die vollständig ausgeführt werden muß, um die logische Konsistenz der gespeicherten Daten zu gewährleisten. Eine logische Transaktion kann eine oder mehrere Aktualisierungsoperationen, die sich auf eine oder mehrere Datenbank-Tabellen beziehen, beinhalten. Darüber hinaus kann eine Transaktion sich auf mehrere NATURAL-Programme erstrecken. Mit einer END TRANSACTION-Anweisung wird eine Transaktion beendet. Diese veranlaßt die Aktualisierung der Datensätze und hebt ihre Sperrung auf. Eine Transaktion kann mit der BACKOUT TRANSAKTION-Anweisung abgebrochen werden. Nach dieser Anweisung werden alle Aktualisierungsoperationen rückgängig gemacht und die gesperrten Datensätze werden freigegeben. Eine logische Transaktion kann auch wiederholt ausgeführt werden. Nach dem erfolglosen Abschluß einer Transaktion können die explizit protokollierten Hauptkommandos mit einer GET TRANSACTION DATA-Anweisung gelesen und erneut ausgeführt werden [Software 93].

Sperrkonzepte

Die Datenbanksätze, die in einer Transaktion verändert werden sollen, müssen gegen konkurrierende UPDATEs gesperrt werden. ADABAS kennt dafür keinen speziellen Befehl. Wenn NATURAL auf die ADABAS-Datenbank zugreift, erfolgt die Sperrung durch Aufnahme der Sätze in eine Holdliste. Die in einer NATURAL-Transaktion aktualisierten Datensätze befinden sich damit automatisch solange in einem *hold*-Zustand bis die Transaktion mit der END-Anweisung erfolgreich beendet oder mit der BACKOUT-Anweisung abgebrochen wird. Die Datensätze im *hold*-Zustand sind für andere Programme nicht verfügbar. Die Transaktionen, die sich um die in *hold*-Zustand gesperrten Datensätze bewerben, müssen in einem Wartezustand (*wait status*) auf Freigabe dieser Daten warten. Um das zu lange Warten auf gesperrte Sätze zu vermeiden, verfügt NATURAL über System-Parameter, die eine explizite frühzeitige Freigabe der Datensätze ermöglichen. Darüber hinaus kann das Verhalten eines Programmes, das auf einen Satz im *hold* wartet, explizit gesteuert werden. So wartet das Programm beispielsweise auf einen gesperrten Satz nicht mehr, wenn der Warte-Parameter auf OFF gesetzt ist.

4.2.7 Programmierung grafischer Benutzerschnittstellen

NATURAL folgt einem *built-in* Ansatz zur Entwicklung grafischer Benutzeroberflächen. Die Möglichkeit zur Darstellung und Verarbeitung grafischer Benutzeroberflächen sind im System integriert. NATURAL erlaubt die interaktive Erstellung von grafischen Benutzeroberflächen. Anschließend wird ihnen einer ereignisabhängiger Programmcode zugewiesen, der auch die Eigenschaften von grafischen Elementen der Oberfläche abfragen und verändern kann [Software 96a]. In diesem Abschnitt wird die Entwicklung grafischer Benutzeroberflächen in NATURAL näher betrachtet.

Die grafische Benutzeroberfläche der ereignisgesteuerten Anwendungen wird in NATURAL mit NATURAL-Dialog-Objekten realisiert. Die Dialoge sind die erste Elemente, die für ereignisgesteuerte Anwendungen implementiert werden müssen. Die Dialog-Objekte sind als Ausgangspunkt der Anwendungen zu verstehen. Eine Anwendung ist gestartet, wenn der Hauptdialog geöffnet wird. Ein Dialog besteht aus einer Eingabemaske und Programmcode. Die Eingabemaske enthält die grafischen Kontrollelemente wie Eingabefelder, Tabellen usw. Das Layout der Eingabemaske und die Eigenschaften der Kontrollelemente werden mit dem Dialog-Editor festgelegt (s. Abschnitt 4.3). Die Funktionalität der Dialog-Objekte wird durch Event-Handler festgelegt. Ein Event-Handler repräsentiert den NATURAL-Code, der beim Auslösen eines Ereignisses ausgeführt werden soll. Der NATURAL-Code kann als Inline-Soubroutine oder Subprogramm realisiert werden (s. Abbildung 4.8).

Die Realisierung der Dialogsteuerungskomponente und Anwendungslogik als Subprogramm ermöglicht die Trennung der Präsentation von der Anwendungslogik und ihre flexible Verteilung im Netz. Drei wichtige Event-Handler des Dialogs bilden Before-Open-Event, After-Open-Event und Close-Event. Im Before-Open-Event kann die Verarbeitung durchgeführt werden, die unmittelbar nach Funktionsaufruf notwendig ist. Außerdem sind alle Variablen zu initialisieren, die mit Attributen von GUI-Elementen verknüpft sind (z.B. Titel des Windows). Im After-Open-Event können Attribute von GUI-Elementen eingestellt werden, die für diese Funktion notwendig sind. Die komplette Endverarbeitung ist im Close-Event vorgesehen. Jeder NATURAL-Dialog hat einen ERROR-Event, der auf Fehlersituation z.B. durch Ausgabe einer Message-Box reagiert. Es wird zwischen folgenden Fenstertypen unterschieden:

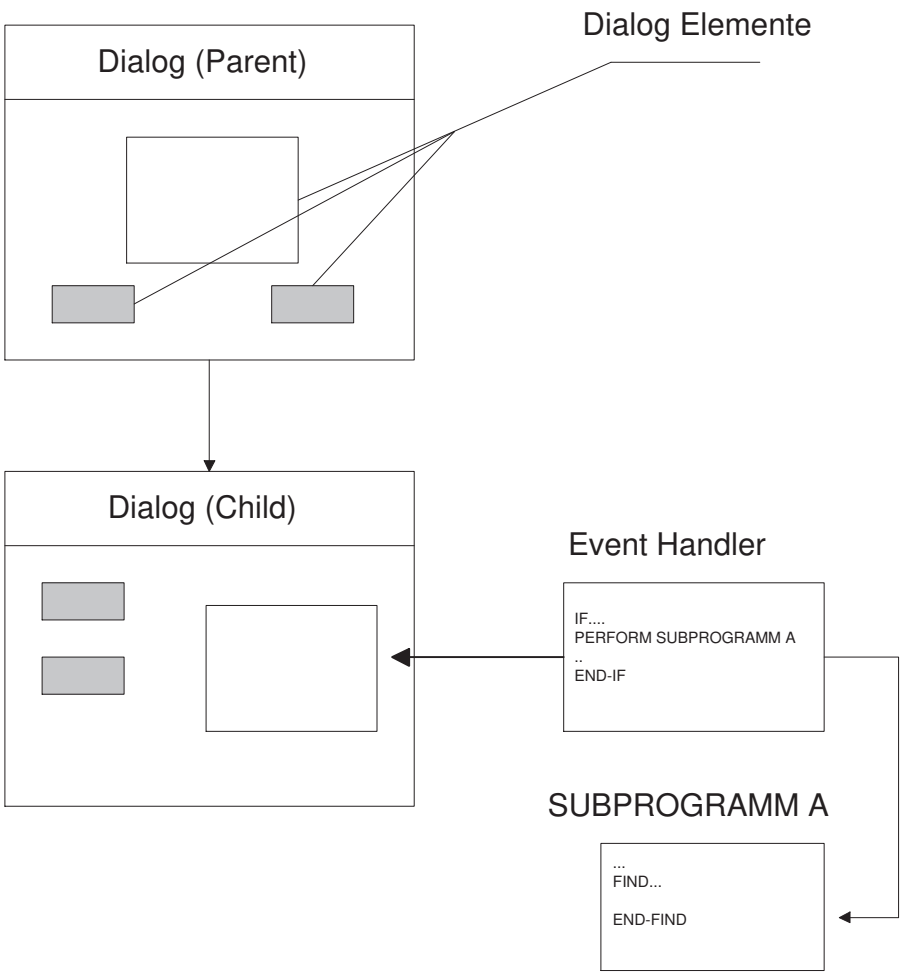

Abbildung 4.8: Dialog-Objekte in NATURAL

- Hauptdialog: Der Einstieg in eine Funktion der Anwendung wird durch den Hauptdialog realisiert, z.B. Anzeige eines Objektes.
- Subdialog: Ein Subdialog wird vom Hauptdialog gestartet und kann für die Darstellung zusätzlicher Attribute eines Objektes verwendet werden. Die Subdialoge werden dem Hauptdialog untergeordnet und stehen in einem engen Zusammenhang zum Hauptdialog.
- Modale Fenster: Modale Fenster können Subdialogen und Hauptdialogen zugeordnet sein. Mit ihnen wird eine Funktionalität realisiert, die eine unmittelbare Reaktion des Endanwenders voraussetzt.

4.3 Entwicklungswerkzeuge

NATURAL bietet eine komplette Produktlinie für die Anwendungsentwicklung. Die Kernkomponente der Entwicklungsumgebung besteht aus

- einem Repository zur Dokumentation und Verwaltung von Daten und Programmen,
- Analyse- und Designwerkzeugen, Programm-, Datenbereich- und Dialog/Maskeneditoren und
- einem Anwendungsgenerator.

Zur Entwicklung großer Mainframe-Anwendungen werden das NATURAL-Anwendungsentwicklungssystem, das Dictionary PREDICT und der Anwendungsgenerator NATURAL-CONSTRUCT eingesetzt. Diese Werkzeuge sind sehr an Eigenschaften der Mainframe-Anwendungen, d.h. zentrale Abläufe und zeichenorientierte Benutzeroberflächen angepaßt. Die Entwicklungsumgebung verfügt über eine menügesteuerte, zeichenorientierte Oberfläche. Erst mit NATURAL New Dimension und seinem Nachfolger NATURAL *Lightstorm* (NLS) ist dem Entwickler eine fensterbasierte Oberfläche zur Verfügung gestellt und die Entwicklung von Client/Server-Anwendungen mit grafischen Oberfläche ermöglicht. NATURAL *lightstorm* verfügt über eine *Engineering Workbench*, einen Mehrbenutzer-Repository, einen Anwendungsgenerator (*Frame Gallery*), und eine Implementierungsumgebung. Die NATURAL-Engineering-Workbench (NEW) unterstützt den Software-Lebenszyklus von der Analyse bis zur Wartung mit visuellen Methoden. Die Entwicklung erfolgt auf Workstations im Einzel- oder Mehrbenutzer-Betrieb. Die NATURAL-Objekte können zwischen dem NATURAL-Repository und NATURAL-PREDICT ausgetauscht werden.

In diesem Abschnitt werden die in NATURAL *LightStorm* für die Entwicklung von Client/Server-Anwendungen relevanten Werkzeuge vorgestellt.

4.3.1 Werkzeuge für die Analyse der Geschäftsprozesse

In NEW werden verschiede Analyseformen unterstützt. Analysiert werden Funktionsstrukturen, Informationsstrukturen und Kommunikationsstrukturen. Die einfache visuelle Analyse der Anforderungen ermöglicht die Zusammenarbeit der DV-Abteilung und der Organisationsabteilungen [Software 96c].

- Funktionsstrukturanalyse: Hier wird der betriebliche Funktionsbaum analysiert. Als Hilfsmittel werden Funktionsbaum und Funktionsnetz-Diagrammtypen eingesetzt.

- Informationsstrukturanalye: Mit dem Entity-Relationship-Diagrammtyp wird die Informationsstruktur abgebildet.
- Kommunikationsstrukturanalyse: Durch den Datenflußdiagrammtyp werden alle Arten von Kommunikation zwischen den internen Funktionen und den fachlich relevanten Informationen untersucht.

Alle bei der Projektarbeit anfallenden Daten und Informationen werden im NATURAL-Repository abgelegt. Außerdem werden zwischen miteinander gekoppelten Objekten sog. Referenzen gebildet. Hat eine Funktion mehrere Teilfunktionen, so werden zwischen der Vaterfunktion und den Söhnen Referenzen im Repository aufgebaut. Eine Funktion im Repository kann von mehreren Diagrammen referenziert werden.

4.3.2 Werkzeuge zur Implementierung

Für die Implementierung der Anwendungen bietet NLS

- Editoren zum Erstellen von Programmen, grafischen Benutzeroberflächen und von Datenbank-Sichten
- Anwendungsgenerator Frame-Gallery zum Generieren von Programmen, grafischen Benutzeroberflächen und Datenbank-Sichten und
- Test-Werkzeuge: Debugger, Call-Statistik, Aufzeichnung von Eingabe- und Bildschirmsequenzen, Protokolle der Datenbankzugriffe.

Diese Werkzeuge sind in einer grafischen Benutzeroberflächen integriert und erlauben eine visuelle Programmierung.

Programmeditor

Der Programmeditor unterstützt die Erstellung und Pflege von Programmen, Unterprogrammen, Subroutinen, Datenbereichen und Copycodes und beinhaltet u.a. folgende Funktionen:

- Zeilen und Blockkommandos zum Löschen, Einfügen, Positionieren, Kopieren, Suchen und Ersetzen von Zeichenfolgen,
- Dynamische Konvertierung von Groß- und Kleinschreibung,
- den *Struct* Befehl zum Strukturieren der eingegebenen Programmsource.
- den *Split-Screen* Befehl zum Einblenden weiterer Programme oder Datendefinitionen.
- den *Syntax coloring* Option zur farbigen Aufbereitung der Source, die das Navigieren und die Fehlersuche erleichtert.

Entwicklungswerkzeug für grafische Benutzerschnittstellen

Der Dialogeditor erzeugt ereignisorientierte Anwendungen mit grafischer Oberfläche. Seine wichtigsten Verarbeitungsfunktionen sind:

- Definiton von Datenbereichen: Mit dieser Funktion können Daten inline deklariert bzw. externe Datenbereiche zum Dialog eingebunden werden.

- Definition von Event Handler: Kodierung des Sourcecodes, der beim Auftreten eines Events ausgeführt werden soll.
- Definition von Inline-Subroutinen: Kodierung des Sourcecodes, der die Steuerung und Verarbeitung im Dialog unabhängig von Events übernimmt.
- Verarbeitung der GUI-Elemente: Die Eigenschaften der GUI-Elemente (List-Box, Scrollbar, Push Button, ..) im Dialog können geändert werden. Darüber hinaus können die GUI-Elemente vergrößert/verkleinert und ausgerichtet werden. Die Ereignisse zur Steuerung der GUI-Elemente werden ebenfalls definiert.

Anwendungsgenerator (Frame Gallery)

Die Frame-Gallery kann man als Programmgenerator von NLS bezeichnen, eine schnelle Entwicklung von GUI-Anwendungen ermöglicht. Mit Hilfe der Frame-Gallery können sowohl Datenbank-Sichten als auch Rahmen für unterschiedliche Dialogtypen generiert werden. In der Frame-Gallery sind Rahmen für das Auflisten, das Auswählen, das Verwalten, die Schlüssselverarbeitung und das Löschen der Daten vorgesehen, womit die Dialogstruktur einer datenintensiven Anwendung zum großen Teil realisierbar ist. Auf der Basis der definierten Datenbank-Sichten wird auch eine Zugriffsschicht komplett generiert, die eine Kapselung aller Datenbankzugriffe, den Einzel-Mehrobjektzugriff und das Transaktionskonzept, beinhaltet [Software 96a].

Repository

Alle Entwicklungsdaten werden im Repository abgelegt. Dies umfaßt alle Objekttypen der Analyse-, Design- und Implementierungsphase. Die im Repository abgelegten Objekte können gesperrt werden, wenn sie geändert werden sollen. Im Einzelbenutzer-Betrieb werden die Daten in der lokalen Datenbank gespeichert. Im Mehrbenutzer-Betrieb werden die Repository-Daten auf einem UNIX-Server verwaltet. Daten können zwischen LAN-Repository und Einzelbenutzer-Betrieb ausgetauscht werden. Jedes Objekt ist unabhängig von der Plattform und dem Repository, auf dem es erzeugt wurde, eindeutig identifizierbar. Werden beim Import-/Export von Daten Objekte mit gleichem Namen gefunden, kann der Benutzer entscheiden, ob das vorhandene Objekt ersetzt werden soll oder nicht.

4.3.3 Software-Management-Werkzeuge

Die Werkzeuge zum Einsatz von Software in verteilten Umgebungen stehen als Bibliotheksdienste zur Verfügung. Diese Dienste verfügen über eine eigene Benutzeroberfläche, die eine einfache Portierung der NATURAL-Anwendung und Verteilung der Modulen ermöglicht.

- Die Dienste der Bibliothek SYSTRANS ermöglichen die Portierung von NATURAL-Sourcen (Programmen, Subprogrammen, Datenbereichen, Masken und Dialogen) zwischen Plattformen. Nicht portiert werden können Programme, die auf NATURAL-Zusatzprodukte bezogene Sprachelemente enthalten, falls diese Zusatzprodukte nicht auf jeder Plattform verfügbar sind. Die Portierung von graphischen in zeichenorientierte Benutzeroberflächen kann sich schwierig gestalten.
- Die Bibliothek SYSRPC stellt Dienste zur Verwaltung von Client- und Server-Komponenten zur Verfügung. Auf der Client-Seite werden die relevanten Stubs ge-

neriert und die Subprogramme, die zu verteilen sind, auf die Server-Plattform transportiert. Die verteilten Subprogramme werden dann als entfernte Dienste in das Serviceverzeichnis eingetragen.

Kapitel 5

System R/3: Eine Entwicklungsumgebung für betriebliche Informationssysteme

Das System R/3 (Realtime-System) basiert auf mehr als 20 Jahren Entwicklungserfahrung der SAP AG auf dem Gebiet betriebswirtschaftlicher Realtime-Anwendungen. Bei der Entwicklung des Systems wurden die folgenden Prinzipien verfolgt [Buck-Emden, Galimow 95]:

- Trennung der Anwendungslösungen von systemnahen Funktionen durch eine klare Schichtenarchitektur des Systems mit einer Anwendungsebene und einer Basisebene (s. Abschnitt 4.1)
- Ganzheitliche Sicht auf alle betrieblichen Vorgänge durch Gewährleistung zusammenhängender betriebswirtschaftlicher Daten über alle Prozeßstufen und Organisationseinheiten hinweg
- Nutzung der Client/Server-Technologie (Die R/3-Architektur basiert auf einem mehrstufigen Client/Server-Konzept mit Methoden zur Steuerung der Dienstnehmer/ Diensterbringer- Beziehung zwischen einzelnen Software-Komponenten.)

Diese Eigenschaften führen zum Einsatz der SAP-Software als De-facto-Standard im Bereich betriebswirtschaftlicher Standardsoftware.

In diesem Kapitel werden die internen Qualitätsmerkmale der Entwicklungsumgebung des Systems R/3 beschrieben. Den Rahmen für die Struktur dieser Beschreibung liefert der im Kapitel 3 aufgestellte Katalog. Im ersten Teil wird die Architektur des Systems beschrieben. Der zweite Teil stellt das Programmiermodell der 4-GL Sprache ABAP/4 (Advanced Business Application Programming) vor. Anschließend werden die Werkzeuge der ABAP/4-Development-Workbench für die Entwicklung von Client/Server-Anwendungen kurz beschrieben.

5.1 Die Architektur des Systems R/3

Der Aufbau des Systems R/3 folgt einem Schichtenmodell, das in Abbildung 5.1 dargestellt ist. Die Hauptschichten dieser Architektur sind:

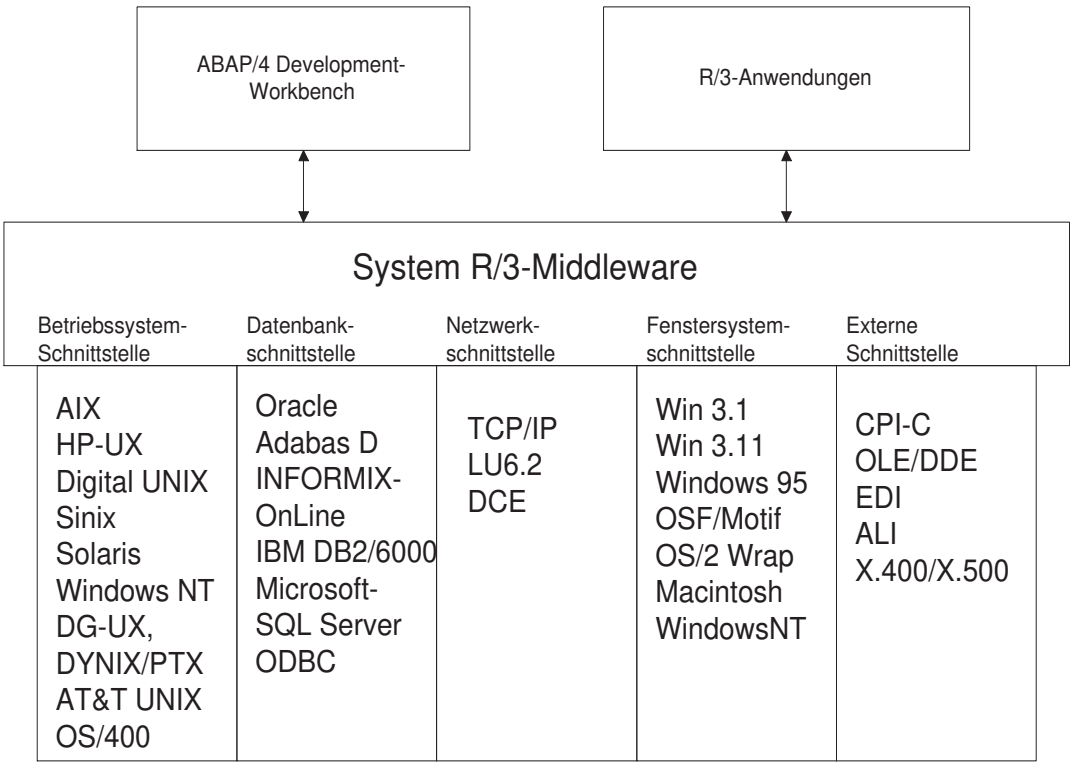

Abbildung 5.1: Schichtenarchitektur des Systems R/3

- Basisschicht: Die Basisschicht umfaßt die Systemdienste und die R/3-Middleware, die in einer heterogenen Umgebung eine homogene Schnittstelle für alle aufsetzenden Dienste und Programme darstellt. Diese Schnittstelle sorgt für die Unabhängigkeit der Anwendungen von den benutzten Systemschnittstellen des Betriebssystems, des Datenbanksystems und des Kommunikationssystems.

- Anwendungsschicht: Die Anwendungsschicht setzt auf der Basisschicht auf. Sie implementiert die betriebswirtschaftlichen Funktionen und Prozesse des Systems R/3 und ist in der 4GL-Sprache ABAP/4 geschrieben.

- ABAP/4-Development-Workbench: Die ABAP/4-Development- Workbench ist die Programmierumgebung. Sie begleitet den gesamten Software-Entwicklungszyklus mit Werkzeugen für die Modellierung, die Programmierung in 4GL-Sprache, den Datenbankzugriff, die Netzwerkkommunikation und die Realisierung von grafischen Benutzerschnittstellen.

Die Basisschicht bietet eine Reihe offener, standardisierter Schnittstellen, die die Portabilität der R/3- Anwendungen bzgl. verschiedenen Betriebssystemen und Datenbanken und ihre Interoperabilität mit externen Anwendungen ermöglicht. Diese Schnittstellen werden nachfolgend näher erläutert.

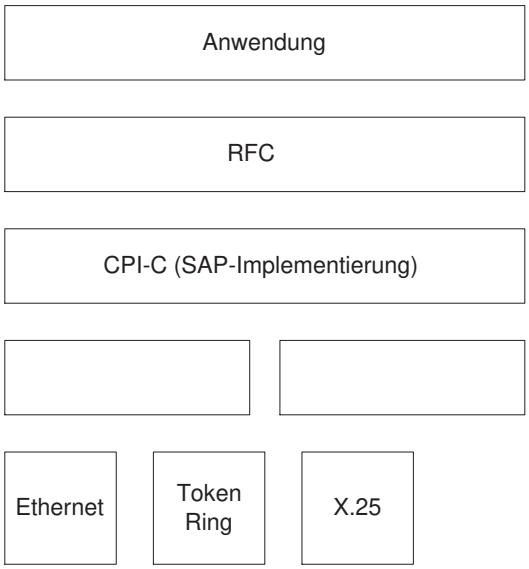

Abbildung 5.2: Kommunikationsprotokollschichten im System R/3

5.1.1 Betriebssystemschnittstelle

Das System R/3 ist auf einer Vielzahl von Plattformen mit vollem Leistungsumfang lauffähig (s. Abbildung 5.1).

5.1.2 Netzwerkschnittstelle

Für die Kommunikation innerhalb des Client/Server-Systems R/3 wird TCP/IP verwendet. Die Anbindung des Systems R/3 an IBM-Großrechner erfolgt über das SNA-Protokoll (Systems Network Architecture) und LU6.2. Eine Implementierung von CPI-C (Common Programming Interface-Communication)[1] sorgt für die Unabhängigkeit der R/3-Anwendungen von den unterliegenden Transportprotokollen.

Das in Abbildung 5.2 dargestellte Protokoll-Schichtenmodell vom System R/3 erlaubt daher die Erweiterung der R/3-Architektur auf weitere Netzwerkprotokolle.

5.1.3 Datenbanksystemschnittstelle

Die Definition und die Manipulation von Daten im System R/3 ist ausschließlich mit SQL-Anweisungen möglich (s. Abschnitt 5.2.7). Die Datenbankschnittstelle des Systems setzt diese Anweisungen für die jeweils verwendete relationale Datenbank um. Damit isoliert sie die Unterschiede in der Syntax und der Semantik der SQL-Implementierung verschiedener Datenbankhersteller. Auf Grund dieser Schnittstelle können prinzipiell alle marktgängigen, relationalen Datenbanksysteme unterstützt werden.

[1] CPI-C ist eine Spezifikation von IBM für die Programm-zu-Programm-Kommunikation. Programme, die CPI-C benutzen, sehen das physikalische Netz nicht.

5.1.4 Schnittstellen zu externen Anwendungen

Die Daten- und Funktionsintegration der R/3-Anwendungen mit externen Anwendungen wird durch folgende Schnittstellen ermöglicht:

- Schnittstelle zur Integration betriebswirtschaftlicher Anwendungssysteme: Die Application Link Enabling (ALE)-Schnittstelle ermöglicht die lose Kopplung betriebswirtschaftlicher Module auf verteilten Systemen mit unterschiedlichen Datenbanken. Zum Beispiel kann ein zentrales Finanzsystem über ALE-Schnittstelle mit dezentralen Logistiksystemen kooperieren und Nachrichten austauschen.
- Schnittstellen zur Integration mit PC-Anwendungen: PC-Anwendungen werden durch folgende Schnittstellen und technische Möglichkeiten mit R/3-Anwendungen verbunden [Buck-Emden, Galimow 95]:
  - Schnittstelle für den File Transfer: Diese Schnittstelle ermöglicht einen bidirektionalen Datenaustausch (Texte und Tabellen) zwischen R/3-Anwendungen und PC-Anwendungen über den File-Transfer-Service.
  - Objektverwaltung mit SAPoffice: Mit dem Mail- und Ablage-System SAPoffice kann das System R/3 PC-Objekte unterschiedlichster Art verwalten, z.B. Arbeitsblätter aus einer Tabellenkalkulation, Texte aus MS-Word, Bilder aus einem optischen Archiv usw.
  - MS-Excel als R/3 Listviewer: Die R/3-Komponente XXL (extended Excel) nutzt MS-Excel als Listviewer. Damit wird die Funktionalität von Excel im System R/3-Umfeld zur Verarbeitung der R/3-Listen verwendet. Ein Benutzer, der mit dem XXL-Listviewer arbeitet, kann die im Listobjekt enthaltenen Daten jederzeit zur individuellen Nutzung in die Standard-Excel Umgebung exportieren. Die Daten sind dann nicht mehr unter der Kontrolle des Systems R/3.
  - Open Data Base Connectivity (ODBC): PC-Anwendungen können auf die Anwendungsdaten direkt in der Datenbank des Systems R/3 über ODBC zugreifen.
  - OLE (Objekt Linking and Embedding) und DDE

5.1.5 Schnittstellen zu anderen Sprachen

Der Remote Function Call (RFC) ist eine High-Level-Programmierschnittstelle zum entfernten Aufruf von R/3-Funktionbausteinen auf anderen Rechner. Die RFC-Bibliothek bietet auch externen Programmen Dienste sowohl zum Aufruf von R/3-Funktionsbausteinen als auch zum Empfang von Funktionsaufrufen aus dem System R/3. Damit stehen den R/3-Funktionbausteinen externe Programme zur Verfügung. Beim Aufruf der externen Programme aus dem System R/3 werden die notwendigen Interface-Programme (z. B. Stub für ANSI-C) von RFC-Schnittstellen-Generator generiert. Durch bidirektionale Gateways ist die RFC-Bibliothek bereits in Programmierumgebungen C, C++, Visual Basic und Tycoon integriert.

5.1.6 Fenstersystemschnittstelle

SAP-GUI (Graphical User Interface) ist die Benutzerschnittstelle des Systems. SAP-GUI bietet eine Untermenge der üblichen Kontrollelemente, die eine einheitliche Präsentati-

on auf unterschiedlichen Desktop-Systemen wie OS/2 Presentation Manager, OSF/Motif, Macintosh und Windows unterstützen. Mit dem Intelligent Terminal API (Application Programming Interface) steht eine weitere Benutzerschnittstelle zur Verfügung. API ermöglicht die kombinierte Darstellung von R/3-Daten und Fremddaten aus einer Großrechneranwendung unter einer gemeinsamen, neuen Oberfläche. Darüber hinaus können fremde Programmierumgebungen auf das Intelligent Terminal API zugreifen. Dieses ermöglicht die Datenaufbereitung entsprechend der zu dieser Programmierumgebung gehörenden Oberfläche. Das Intelligent Terminal API steht auf Windows NT und unter Windows 95 zur Verfügung.

5.2 Das ABAP/4-Programmiermodell

Für alle Programmentwicklungen im R/3-Anwendungsbereich wird die 4GL- Programmiersprache ABAP/4 benutzt. Die folgende Beschreibung dieser Sprache gibt einen Überblick über ihre Historie sowie softwaretechnische Charakteristika:

Ausgehend von den ersten Versionen im System R/2 wurde und wird die Programmiersprache ABAP/4 beständig weiterentwickelt. Jede neue Version beinhaltet neue Merkmale und Konzepte. Da die Entfernung veralteter Bestandteile aus Gründen der Kompatibilität nicht immer möglich ist, wird durch das Einfügen neuer Konzepte der Umfang der Sprache immer größer und unübersichtlicher.

Beim Übergang auf grafische Benutzeroberflächen und Realisierung einer Client/Server-Architektur, mußte die Programmiersprache flexibler werden. Deshalb wurden die bestehenden Kommandos mit neuen Optionen erweitert, wodurch einige Kommandos eine völlig neue Funktionalität erhalten.

Es existiert kein Standard, in dem der Sprachumfang von ABAP/4 beschrieben wurde. Die Sprache ist ihr eigener Standard. [Matzke 96]

In diesem Abschnitt wird auf die Konzepte des APAP/4-Programmiermodells eingegangen. Diese Beschreibung basiert zum großen Teil auf SAP-eigenen Publikationen.

5.2.1 Datenstrukturierungskonzepte

Die Datentypen von ABAP/4 orientieren sich an den Anforderungen betriebwirtschaftlicher Anwendungen. Diese Orientierung zeichnet sich durch vordefinierte Datentypen wie Datum und Zeitpunkt und eine Reihe von Operationen zur Verarbeitung von Textdaten aus.

Neben den vordefinierten Datentypen ist ein Typkonzept zur Benennung der Datentypen und ihre Wiederverwendung vorhanden. Die zusammengesetzten Datentypen wie Feldleisten und interne Tabellen erlauben die Aufnahme komplexer umfangreicher Zwischenergebnisse. Diese Datentypen können ebenso als Aufrufparameter bei einem entfernten Funktionsaufruf verwendet werden. Es existieren keine Einschränkung bzgl. verwendeten Plattformen für die Benutzung von Datentypen in verteilten Anwendungen.

Vordefinierte Datentypen

Neben den klassischen Datentypen wie ganze Zahlen, Fließkommazahlen, Zeichenketten usw. sind die für beriebswirtschaftliche Anwendungen sehr relevanten Datentypen wie Datum und Zeitpunkt in der Sprache vordefiniert. In gemischten Ausdrücken werden durch automatische Konvertierungen Zahlen als Längen von Zeitintervallen interpretiert. Dies

ermöglicht die Addition einer Zahl zu einem Datum. Außerdem existieren umfangreiche Funktionen für die Verarbeitung textueller Daten. Dazu gehören das Suchen und Ersetzen einer vorgegeben Zeichenkette, das Eliminieren überflüssiger Leerzeichen, das Verschieben einer Zeichenkette und das Überprüfen, ob Zeichen aus einer Zeichenkette in einer anderen vorkommen. Die Typumwandlungen zwischen allen vordefinierten Datentypen wird automatisch unterstützt. Zur Unterstützung piktografischer Sprachen wie japanisch und chinesisch ist der Datentyp *Wide Character-Double Byte Character Set* eingeführt. Double-Byte-Zeichen können in einem Zeichenfeld gemischt mit normalen Ein-Byte-Zeichen vorkommen und bearbeitet werden.

Typkonstruktoren

ABAP/4 verfügt über nur zwei strukturierte Datentypen, d.h. die Feldleiste und die interne Tabelle[2]. Die Feldleiste ist ein mit Record vergleichbare Typkonstruktor. Der Begriff Array existiert unter ABAP/4 nicht. Als Ausgleich dazu bietet ABAP/4 die interne Tabellen. Die Eigenschaften dieser Typkonstruktoren werden in diesem Abschnitt erläutert.

Feldleisten

Feldleisten aggregieren mehrere Datenfelder, die von verschiedenen Datentypen sein können, unter einem einzigen Namen. Feldleisten können wie folgt definiert und verwendet werden:

- Mit den Schlüsselwörtern DATA oder TYPES können Feldleistenvariablen bzw. Feldleistentypen angelegt werden.
- Feldleisten können nur Feldleisten und Datenfelder von elementaren Datentypen enthalten. Es ist nicht möglich, Funktionen oder Tabellen als Komponente einer Feldleiste zu definieren.
- Die Anweisung LIKE ermöglicht einem Datenfeld die Vererbung der Eigenschaften eines anderen Datenfeldes, wie z.B. Typ eines Tabellenfeldes. Die vererbten Eigenschaften werden einem Datenfeld zur Laufzeit des Programms zugewiesen.
- Beim Zugriff auf Feldleisten ist zwischen einem Zugriff auf die Feldleiste als Ganzes, auf einem Teil und auf einzelne Felder zu unterscheiden.

Interne Tabellen (smart Arrays)

Die einzige Möglichkeit, mehrere gleichartige Datensätze zusammenzufassen sind interne Tabellen. Interne Tabellen ähneln Feldleisten, bestehen aber aus mehreren Feldleisten. Die internen Tabellen dienen innerhalb einer Anwendung der temporären Datenspeicherung und Bearbeitung, z.B. zum Datenaustausch mit Unterprogrammen oder Funktionsbausteinen.

Die Basis-Operationen der internen Tabellen sind:

- Definition interner Tabelle: Interne Tabellen werden wie Feldleisten innerhalb eines Blocks definiert. Es existiert keine Obergrenze für die Kardinalität der Tabelle. Bei Bedarf werden interne Tabellen vom System dynamisch erweitert. Es können auch

[2] Ältere Releasestände der R/3-Software (kleiner als 3.0) verfügen nicht über das Typkonzept. Daher ist die Möglichkeit Feldleisten anzulegen, eingeschränkt.

Typanweisungen für eine interne Tabelle erzeugt werden. Bezüge mit LIKE für die Deklaration des Datensatzes sind erlaubt.

- Lesen, Schreiben und Löschen in internen Tabellen: In ABAP/4 existieren vordefinierte Operationen, wie z.B. DELETE, INSERT, MODIFY, zur Verarbeitung der Tabellen, wie z.B. das Löschen, das Einfügen und das Ändern von Zeilen. Der zu bearbeitende Datensatz wird über seinem Index angesprochen. Weiterhin erlaubt die Operation LOOP eine Iteration über alle Zeilen einer internen Tabelle.

Benutzerdefinierte Datentypen

Mit Release 3.0 ist ABAP/4 um ein Typkonzept erweitert, das die Benennung benutzerdefinierter Datentypen erlaubt. Die mit TYPES definierten Datentypen können in anderen Definitionen ebenso wie die vordefinierten Typen benutzt werden. Die Typdeklarationen werden in Typ-Pools ausgelagert, die in verschiedene Anwendungen eingebunden werden können. Bei der Deklaration von Typen ist auch die Bezugnahme auf bereits existierende Typen mit der LIKE-Anweisung möglich.

5.2.2 Strukturierungstechniken für die Programmierung im Kleinen

In diesem Abschnitt werden die Möglichkeiten der Sprache ABAP/4 zur Beschreibung von imprativen Algorithmen dargestellt.

Blockstrukturierte Kontrollstrukturen

In der Sprache ABAP/4, deren Syntax COBOL und Pascal sehr ähnlich ist, existieren Steuerungs-Sprachelemente zur Realisierung von Kontrollstrukturen. Neben Kontrollsturkturen (IF ... ENDIF, CASE ... ENDIF) zur bedingten Ausführung gibt es verschiedene Arten von Schleifen mit unterschiedlichen Abbruchbedingungen (DO ... ENDDO, WHILE ... ENDWHILE, LOOP ... ENDLOOP). Diese Kontrollstrukturen unterstützen sowohl COBOL als auch PASCAL-Syntax für Schleifen. Sie erlauben die flexible Realisierung von Iterationen über die Tabellen.

Prozedurale Abstraktion

Um den Quelltext der Programme übersichtlicher zu halten und die mehrfache Nutzung von Programmteile zu ermöglichen, benutzt ABAP/4 Unterprogramme. Diese ähneln den von Pascal bekannten Prozeduren. Sie können Parameter entgegennehmen, geben aber keinen Funktionswert zurück. Unterprogramme gestatten es, Anwendungen in kleinere, abgeschlossene Teile zu gliedern. Sie können in einem FROM ... ENDFROM-Block in jedem Typ von Programm definiert werden. Unterprogramme können sich in einem anderen als dem aufrufenden Programm befinden. Sowohl der Name des zu rufenden Unterprogramms als auch der Name des Programms, in dem das Unterprogramm steht, sind beim Aufruf dynamisch zu bestimmen. Unterprogramme dürfen lokale Felder enthalten, die anderen Programmen nicht zugänglich sind. Parameter können wahlweise als Werte (*call by value*) oder als Referenzen (*call by reference*) an Unterprogramme übergeben werden. Ein entfernter Aufruf eines Unterprogramms ist nicht möglich.

Ausnahmebehandlung

Eine ABAP/4-Anwendung kann beim Auftreten unvorhergesehener Ereignisse individuelle Fehlermeldungen (im System R/3 als Nachrichten bezeichnet) ausgeben. Da die Anzeige von Nachrichten Einfluß auf den Programmablauf hat, werden die Fehler innerhalb von Funktionsbausteinen durch ein Ausnahmekonzept behandelt. Die Funktionsbausteine lösen keine Nachrichten, sondern sogenannte Ausnahmen aus, welche den Funktionsbausteine beenden und dem aufrufenden Programm einen Fehlercode übermitteln. Im nächsten Abschnitt werden die Nachrichtenkonzepte und Ausnahmen zur Fehlerbehandlung in ABAP/4 näher erläutert.

Nachrichten

Die Fehlersignalisierung im System R/3 erfolgt über die sogenannten Nachrichten. R/3-Nachrichten sind vordefinierte Textelemente. Sie bestehen aus einem einzeiligen Kurztext und einem optionalen, mehrzeiligen Langtext. Das Auslösen von Nachrichten hat Einfluß auf den Programmablauf, worauf der Programmierer nur beschränkt Einfluß nehmen kann. Die Nachrichten können auf folgende, verschiedene Arten angezeigt werden:

- Die Nachrichten werden in Form eines Kurztextes in der Statuszeile dargestellt.
- Im Dynpro (s. Abschnitt 5.2.8) werden die Felder, auf die sich die fehlerhafte Eingabe bezieht, markiert. Der Cursor steht im ersten zu korrigierenden Feld.
- Die Nachrichten werden in einem eigenen Fenster angezeigt. Nach dem Schließen des Fensters wird das Programm fortgesetzt.

Ausnahmen

Die Funktionsbausteine signalisieren durch Ausnahmen Fehlerzustände. Damit sind nicht nur Laufzeitfehler gemeint. Sondern Ausnahmen können auch ausgelöst werden, wenn z.B. ein Funktionsbaustein einen Datensatz nicht finden kann.

Beim Erzeugen eines Funktionsbausteins werden die Bezeichner für die Ausnahme, die innerhalb des Funktionsbausteins erzeugt werden sollen, in der Funktionsschnittstelle definiert. Beim Erzeugen einer Ausnahme wird einer der Bezeichner als Parameter an das entsprechende Ausnahmekommando übergeben. Soll die Ausnahme durch den Funktionsbaustein selbst behandelt werden, dann erscheint der Bezeichner in Fehlertext.

Sollen Ausnahmen durch ein aufrufendes Programm behandelt werden, werden dem Bezeichner der möglichen Ausnahmen, numerische Werte zugeordnet. Beim Erzeugen einer Ausnahme wird der zugehörige numerische Wert in eine Systemvariable gestellt. Das rufende Programm kann anhand dieser Variablen die Ausnahmesituation behandeln.

Multi-Threading

Die Ausführung von Transaktionen (s. Abschnitt 5.2.7) ist ein mit Threads vergleichbares Konzept im System R/3. Eine Transaktion ist eine Folge logisch zusammenhängender Programmschritte. Die Ausführung einer Transaktion muß nicht innerhalb eines einzigen Systemprozesses abgewickelt werden. Vielmehr können die einzelnen Dialogschritte einer Transaktion von einem oder von mehreren Prozessen, die als Workprozesse bezeichnet werden, verarbeitet werden. Außerdem besteht die Möglichkeit neue Transaktionen

zu starten, die unabhängig von der Original-Transaktion ausgeführt werden können. Somit können beliebige Hintergrundverabeitungen der Programme parallel zu ihrer online Dialogverarbeitung gefahren werden.

5.2.3 Strukturierungstechniken für die Programmierung im Großen

R/3-Anwendungen weisen eine erhebliche Komplexität auf. Dies erfordert Methoden zur Modularisierung und Wiederverwendung des Quellcodes innerhalb einer Anwendung. In diesem Abschnitt werden die in ABAP/4 vorgesehenden Konzepte zur Modularisierung und Wiederverwendung von Code näher erläutert.

Modularisierungskonzepte

Für die Modularisierung von Programmen kennt ABAP/4 die Definition und den Aufruf der Dialog- und Funktionsbausteine. Sie können unabhängig von einer konkreten Anwendung entworfen und getestet werden. Beispiele sind Funktionen zur Kalenderrechnung oder Dynpros zum Anzeigen oder Erfassen von Standarddaten wie z.B. Adressen. Sie verfügen über eine genau beschriebene Schnittstelle zum Datenaustausch. Eine geschickte Aufteilung der Funktionalität einer Anwendung auf Funktionsbausteine kann eine objektbezogene Programmierung ermöglichen.

Funktionsbausteine

Funktionsbausteine bilden im Wesentlichen die Grundlage für die Modularisierung von ABAP/4-Programmen. Sie zeichnen sich durch die Möglichkeit zur Wiederverwendung von Programmteilen und zur Datenkapselung aus. Funktionsbausteine besitzen eine abgegrenzte Datenschnittstelle, welche die erforderlichen Parameter beinhaltet. Nur über diese Datenschnittstellen sind die Funktionsbausteine ansprechbar. In der Datenschnittstelle eines Funktionsbausteins werden die erforderlichen Parameter, getrennt nach Export-, Import- und Tabellenparametern, beschrieben. Über Tabellenparameter können im Unterschied zu den anderen Parametern, deren Übergabe stets *call by value* erfolgt, Referenzen auf komplette Tabellen übergeben werden. Funktionsbausteine können in einer Funktionssammlung gruppiert werden. Innerhalb einer Funktionsgruppe können Funktionsbausteine auf gemeinsame Daten zugreifen, die nach außen nicht sichtbar sind. Die Funktionsbausteine werden in einer Funktionsbibliothek verwaltet. Diese Bibliothek ist im R/3-Standard bereits mit einer Reihe von Funktionsbausteinen gefüllt. Der entfernte Aufruf der Funktionsbausteine im Netz erfolgt über *Remot Function Call* (RFC) (s. Abschnitt 5.2.4)

Dialogbausteine

Dialogbausteine sind ein weiters Modularisierungskonzept in ABAP/4. Sie ermöglichen die Wiederverwendung von Dynpros. Dialogbausteine umfassen einen Modul-Pool, Dynpros und eine Oberfläche. Der Aufruf der Dialogbausteine geschieht über eine fest definierte Schnittstelle.

Wiederverwendung von Code

Außer der erwähnten Modularisiungkonzepten, d.h. den in Bibliotheken verwalteten Funktions- und Dialogbausteinen, existieren in ABAP/4 nur Include-Programme (Kopierstrecken)

zur Wiederverwendung des Programmcodes. Include-Programme enthalten keinen eigenständig ablaufenden Code. Sie dienen dazu, große Programmeinheiten in kleine, wiederverwendbare Teile zu gliedern. Beim Generieren des Programmcodes oder beim Syntaxcheck erfolgt dann eine rein textuelle Ersetzung der Include-Anweisung durch das Include-Programm.

5.2.4 Verteilungsabstraktion

Die Realisierung der mehrstufigen Architektur des Systems R/3 wird durch die Entkopplung der Anwendungslogik von der Präsentation und der Datenbank ermöglicht. Diese Einheiten können auf verschiedene Rechner gemäß der geplanten Client/Server-Architektur (zweistufig oder dreistufig) verteilt werden. Die Daten eines Datenbank-Servers stehen vielen Anwendungs-Servern parallel zur Verfügung. Anwendungen auf verschiedenen Anwendungs-Servern tauschen Informationen aus und können die Informationen auf einen entfernten Präsentations-Client aufbereiten.

Aufsetzend auf dem in der Middleware verwendeten TCP/IP Transportprotokoll wird das SAP-Präsentationsprotokoll für den Datenaustausch zwischen Präsentations-Client und Anwendungs-Server sowie das Remote-SQL-Protokoll des jeweiligen Datenbankherstellers für den Datentransfer zwischen Applikation- und Datenbank-Server benutzt. Diese Protokolle bilden die Verteilungskonzepte des Systems R/3, die in diesem Abschnitt erläutert werden.

Entfernter Datenbankzugriff

Das Remote-SQL-Protokoll innerhalb der Datenbankschnittstelle des Systems R/3 ermöglicht den entfernten Datenbankzugriff zwischen einer Anwendung und einer entfernten Datenbank. Mit *Remote SQL* wird der Zugriff auf relationale Datenbanken auch in verteilten Client/Server-Umgebungen möglich. Datenbankzugriffe werden vom Datenbanksystem völlig transparent für die Anwendung vom Client zum Server übertragen.

Wegen des hohen Datentransfervolums ist ein entfernter Datenzugriff nur über ein LAN sinnvoll. Ein besonderer Bestandteil der R/3-Datendienste ist die Optimierung des Client/Server-Betriebs. Die Datenbankdienste stellen Puffer im Shared-Memory-Bereich zur Verfügung. Diese Puffer speichern von der Anwendung häufig benötigte Daten im Hauptspeicher, so daß für das Lesen kein Zugriff mehr auf den Datenbank-Server notwendig ist [Buck-Emden, Galimow 95].

Entfernter Prozeduraufruf

Das Prinzip den RPC (*Remote Procedure Call*) in 4GL ABAP/4 wird mit dem RFC (*Remote Function Call*) realisiert. Über RFC erfolgt der Aufruf eines Funktionsbausteins auf einem entfernten Rechner. Für das aufrufende Programm bleibt die Verteilung auf mehreren Systemen vollständig transparent. Allgemein gilt, daß als RFC-Kommunikationspartner R/3-, R/2 und auch externe Anwendungen in Frage kommen können.

Wesentliche Eigenschaften des RFC sind:

- Syntax und Verhalten des RFC entspricht dem Aufruf eines normalen Funktionsbausteins.
- Synchrone und asynchrone Aufrufe sind möglich.

- Der Aufruf erfolgt dynamisch. Eine Generierung von statischen Kommunikationsmodulen (Stubs) ist in einer reinen ABAP/4-Umgebung nicht notwendig.
- Es besteht die Möglichkeit zur individuellen Fehlerbehandlung über einen *Exception* Mechanismus.
- Als Aufrufparameter können alle ABAP/4-Datentypen wie Einzelfelder, Strukturen und Tabellen verwendet werden. Es können jedoch keine Funktionen als Aufrufparameter definiert werden.
- Mit der Call-Back-Funktion kann ein per RFC aufgerufenes Programm seinerseits einen Funktionsbaustein im aufrufenden System starten.
- Die Ausführung der RFCs erfolgt in derselben Reihenfolge und im selben Programmkontext im Zielsystem, wie sie vom aufrufenden System gesendet wird.
- Ein RFC-Aufruf wird nur einmal ausgeführt.

5.2.5 Persistenzabstraktion

In ABAP/4-Programmen werden sowohl langlebige als auch kurzlebige Daten bearbeitet. Die Sprache unterstützt keine Persistenzabstraktion. Die einzige langlebige Datenstruktur ist die transparente Tabelle (s. Abschnitt 5.2.6), die im Data-Dictionary definiert wird. Die Feldleiste und interne Tabelle sind kurzlebige Datenstrukturen und können nicht persistent werden. Zur Verarbeitung von persistenten Tabellen und flüchtigen Tabellen sind unterschiedliche Anweisungen und Operationen in der Sprache vorgesehen. Es ist nicht möglich Algorithmen zu schreiben, die uniform auf persistenten und flüchtigen Tabellen operieren.

5.2.6 Massendatenverarbeitung

Das System R/3 kann als Datenbank ausschließlich relationale Datenbanksysteme verwenden. Daher orientieren sich die Sprachelemente zur Massendatenverarbeitung in ABAP/4 an relationalen Datenmodellen. Die transparenten Tabellen sind die einzige Datenstruktur zur Verarbeitung persistenter Massendaten. Tabellen und Views auf diese Tabellen können aber im ABAP/4-Dictionary definiert werden.

Die Sprache ABAP/4 verfügt nur über SQL-Sprachbefehle zur Manipulation von Daten der Tabellen. Nachfolgend werden die transparenten Tabellen und ihre Definition im Dictionary behandelt. Danach werden die Sprachelemente zur Datenmanipulation in ABAP/4 vorgestellt.

Sprachelemente zur Massendatendefinition

ABAP/4-Programme greifen nicht direkt auf die Datenbestände eines relationalen Datenbanksystems zu. Das im ABAP/4-Dictionary definierte relationale Datenmodell ermöglicht ABAP/4-Programmen den Zugriff auf die unterliegende Datenbank. Um eine konsistente Verbindung sicherzustellen werden die Tabellendefinition und Tabellenmodifikation automatisch generiert und durchgeführt. Die im Dictionary definierten Tabellen werden dabei 1:1 auf das unterliegende Datenbanksystem abgebildet. Die jeweiligen Tabellen heißen daher transparente Tabellen. Aus dem Dictionary heraus können Datenbanktabellen und Datenbank-Views angelegt, gelöscht und modifiziert werden. Diese Realisierung der

Verbindung zwischen ABAP/4-Tabellen und den Datenbankdaten durch das ABAP/4-Dictionary hat die Unabhängigkeit der ABAP/4-Programme von Änderungen in der Organisation der Daten zur Folge.

Transparente Tabellen

Tabellen sind die wichtigste Datenstruktur im System R/3. Sie bestehen aus Feldern. Tabellendefinitionen im ABAP/4-Dictionary geben den Tabellenfeldern Namen und spezifizieren für jedes Feld Eigenschaften wie Format und Länge. Außerdem legen sie die Schlüsselfelder der Tabellen und die Beziehungen zwischen Tabellen fest.

Bei der Beschreibung der Eigenschaften der Tabellenfelder wird zwischen technischen Eigenschaften und betriebswirtschaftlichen Eigenschaften (Metadaten) unterschieden. Die technischen Eigenschaften wie Datentyp, Länge und Bildschirmdarstellung. werden durch Domänen spezifiziert. Zur Beschreibung der Metadaten wie Überschriften, Dokumentationstext und Domänen werden Datenelemente benutzt. Domäne und Datenelemente sind als eigenständige Objekte im ABAP/4-Dictionary definiert. Eine Domäne kann in beliebig vielen Datenelementen vorkommen. Ein Datenelement kann für mehrere Felder wiederverwendet werden.

Eine weitere Möglichkeit zur Erhöhung des Wiederverwendungsgrads ist die Einbettung ganzer Datenstruktur mittels eines Include-Verfahrens in Tabellen. Die Änderung dieser Datenstruktur hat somit die automatische Änderung aller betroffenen Tabellen zur Folge.

Externe Tabellen werden mit der Anweisung TABLES (Tabellenname) in einem ABAP/4-Programm deklariert. Voraussetzung für eine Deklaration mit TABLES ist, daß die Tabelle aktiviert [3] und im Dictionary enthalten sein muß. Die in ABAP/4 integrierten *Open SQL* Sprachmittel und *Native SQL* Befehle unterstützen lesende und verändernde Zugriffe auf diese Tabellen.

Views

Neben Tabellen kann eine Selektion von Daten in der Datenbank auch über Views erfolgen. Nur die Pflege von Daten über Views ist eingeschränkt. In ABAP/4 sind Views als virtuelle Tabellen definiert, die keine eigenen Daten enthalten, sondern eine anwendungsspezifische Sicht auf die im ABAP/4-Dictionary verzeichnete Tabellen bilden. Die Felder aus einer oder mehreren Tabellen werden miteinander verknüpft und dem Benutzer als eine einzige Tabelle dargeboten. Es existieren einige unterschiedliche Viewtypen für verschiedene Anwendungen:

- Datenbank-Views werden als echte Views des Datenbanksystems realisiert und können nur transparente Tabellen aufnehmen.
- Customizing-Views werden beim Customizing eingesetzt und ermöglichen das Editieren von Datensätzen.
- Streich-Views bieten eine Sicht auf eine einzige Tabelle.
- Help-View sind für das Hilfesystem relevant.

[3] Wichtig bei allen Definitionen im Data Dictionary ist, daß ein Objekt erst dann sichtbar wird, wenn es aktiviert wurde.

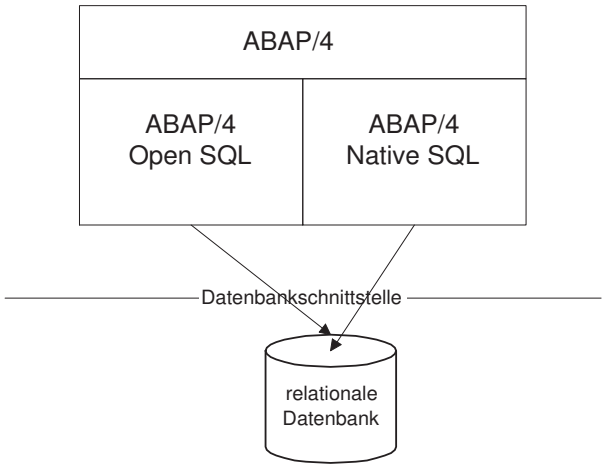

Abbildung 5.3: Sprachelemente zur Datenmanipulation in ABAP/4

- Entitäts-Views sind für das Unternehmensdatenmodell [4] wichtig.

Sprachelemente zur Datenmanipulation

Um die unterschiedliche SQL-Funktionalität in relationalen Datenbanksystemen zu unterstützen, werden zwei SQL-Levels zur Verfügung gestellt (s. Abbildung 5.3):

- ABAP/4 Open SQL: Die in ABAP/4 integrierten *Open SQL* Sprachmittel unterstützen lesende und modifizierende Zugriffe auf transparente Tabellen. *Open SQL* besteht aus einer Untermenge der durch die ANSI [5] standardisierten SQL-Befehle. Alle Anwendungen, die nur über diese Sprachmittel auf die Datenbank zugreifen, sind ohne jede Änderung auf allen vom System R/3 unterstützten Datenbanksystemen ablauffähig. Die Datenbankschnittstelle des R/3-Laufzeitsystems setzt Open-SQL-Anweisungen aus ABAP/4-Programmen in Anweisungen für die jeweils verwendete relationale Datenbank um. ABAP/4-Programme müssen daher weder Besonderheiten der Speicherung noch Inkompatibilitäten zwischen verschiedenen Datenbanksystemen berücksichtigen.

- ABAP/4 Native SQL: Der mit ABAP/4 *Native SQL* verfügbare Sprachumfang erlaubt die Nutzung der vollen Funktionalität des jeweiligen Datenbanksystems. Die *Native SQL* Befehle werden uninterpretiert an die jeweilige Datenbank weitergeleitet. Deswegen werden Programmierfehler in *Native SQL* Anweisungen erst zur Laufzeit erkannt.

  Native SQL ist nicht zwischen verschiedenen Datenbanken portabel. Diese Eigenschaft beschränkt die Verwendung dieser Anweisungen bei der Programmierung auf Anwendungen, die Anfragen an mehrere heterogene Datenbanken realisieren müssen. Auf der anderen Seite können mit diesen Anweisungen die datenbank- und herstellerspezifischen Erweiterungen von Standard-SQL benutzt werden.

[4] Das Unternehmensdatenmodell beschreibt die betriebswirtschaftliche Aspekte der Daten.
[5] Bereits 1986 definiert das ANSI einen ersten Sprachstandard für SQL [Date 89].

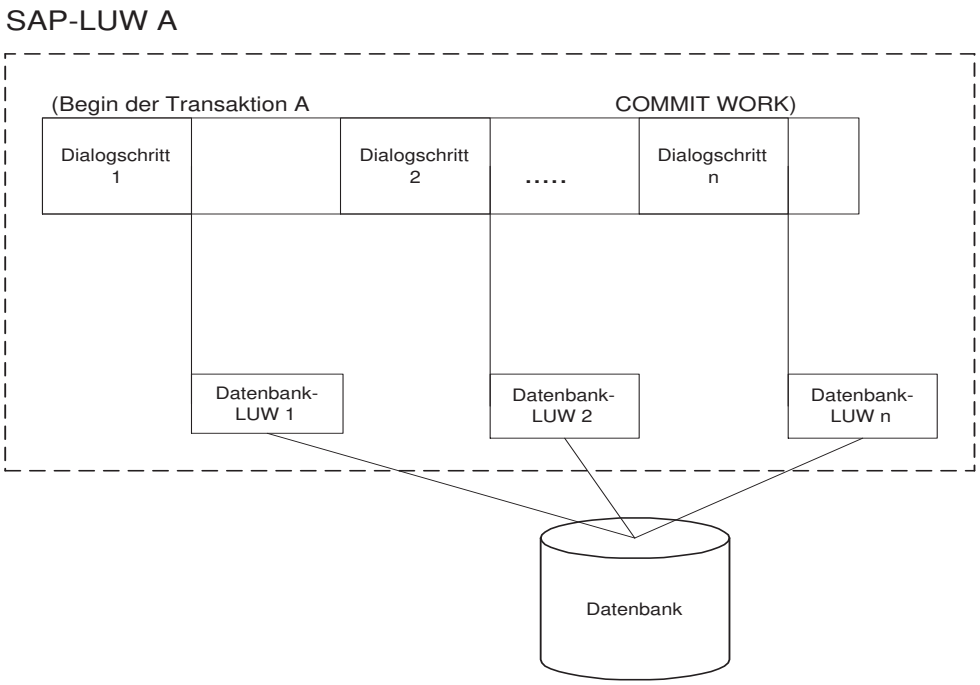

Abbildung 5.4: Transaktionen im System R/3

5.2.7 Transaktionen

Im System R/3 wird zwischen betriebswirtschaftlichen SAP-Transaktionen, SAP-LUW (*logical Units of Work*) und Datenbank-LUW unterschieden. Eine Folge von logisch zusammenhängender Dialogschritte werden als SAP-Transaktionen (anwendungsorientierte Transaktionen) definiert. Die Gesamtheit der Dialogschritte einer Transaktion zuzüglich der Fortschreibung der Datenbank wird mit dem systemorientierten Begriff SAP-LUW bezeichnet. Datenbank-LUWs entsprechen dem von Datenbanken bekannten Transaktionskonzept. Sie dienen der atomaren und dauerhaften Durchführung von Datenbankänderungen. Datenbank-LUWs werden vom System R/3 automatisch bei jedem Bildschirm-Wechsel abgeschlossen. Der Zusammenhang zwischen SAP-Transaktionen, SAP-LUWs und Datenbank-LUWs ist in Abbildung 5.4 dargestellt.

Im nächsten Abschnitt sind diese Konzepte und ihren Zusammenhang beschrieben.

SAP-Transaktionen

Der Kern der Programmabarbeitung bilden die anwendungsorientierten Transaktionen. Ausführbare Anwendungen werden für den Anwender erst zugänglich, wenn für sie eine Transaktion identifiziert wurde. Damit sind Transaktionen die einzigen unmittelbar durch den Anwender ausführbare Einheiten. Eine SAP-Transaktion ist eine Folge betriebswirtschaftlicher zusammenhängender Dialogschritte. Jede Transaktion wird durch einen vierstelligen Transaktionscode identifiziert und kann auch geschachtelt auftreten. Eine

SAP-Transaktion startet immer mit einer eigenen SAP-LUW. Der Aufruf CALL TRANSAKTION erzeugt eine neue SAP-LUW. Diese kann dann mit dem Open-SQL Befehlen COMMIT WORK oder ROLLBACK WORK entweder abgeschlossen oder abgebrochen werden.

SAP-LUWs

SAP-LUWs kapselen datenverändernde Anweisungssequenzen, die aus Open-SQL Befehlen bestehen. Alle datenbankänderenden Open-SQL Anweisungen erzeugen Änderungsaufträge innerhalb der aktuellen SAP-LUW. Die Datenbankänderungen werden in SAP-Tabllen zwischengespeichert. Nach einer COMMIT WORK-Anweisung wird der Änderungsauftrag durch einen entsprechenden Workprozeß ausgeführt und es wird automatisch für die unterliegende Datenbank-LUW ein Commit ausgelöst.

Alle Änderungen, die innerhalb einer SAP-LUW veranlaßt werden, sind mit ROLLBACK-WORK rücksetzbar. Die SAP-LUW ist innerhalb des Systems R/3 die sichere Aktion bezüglich Atomarität und Dauerhaftigkeit. Eine Isolation zu anderen parallel ablaufenden SAP-LUWs muß über Sperrobjekte explizit realisiert werden.

LUWs werden nicht notwendigerweise innerhalb eines einzigen Systemprozesses abgewickelt. Vielmehr können die einzelnen Dialogschritte einer Transaktion von einem oder von mehreren Prozessen, die als Workprozesse bezeichnet werden, verarbeitet werden. Außerdem steht mit der asynchronen Verbuchung ein Verfahren zur Verfügung, mit dessen Hilfe der Dialogteil einer Transaktion und die zugehörige Datenbankfortschreibung in unterschiedlichen Workprozessen oder sogar auf unterschiedlichen Rechner ablaufen können.

Datenbank-LUWs

Eine Datenbank-LUW entspricht einer flachen Datenbanktransaktion mit abschließendem Commit [Lutz 97]. Datenbank-LUWs werden vom System R/3 automatisch bei jedem Bildschirm-Wechsel mit einem COMMIT abgeschlossen. Damit sind die mit Native-SQL-Anweisungen durchgeführte Datenbankänderungen spätestens nach einem Dialogschritt persistent.

Sperrkonzepte

Ein Anwendungsprogramm kann explizit das Sperren von Objekten anfordern. Damit besitzt das Anwendungsprogramm für die Dauer einer Transaktion ausschließlicher Zugriff auf ein bestimmtes Objekt. Fordert ein Anwendungsprogramm Sperren für Datenobjekte an, die sie verändern möchte, werden zuerst die konkurrierenden Datenbankzugriffe bereits im Vorfeld erkannt. Damit kann das Anwendungsprogramm den Benutzer frühzeitig darauf hinweisen, daß die von ihm gewünschte Operation zur Zeit nicht möglich ist. Das Sperren und Freigeben von Datensätzen erfolgt nicht direkt durch spezielle ABAP/4-Kommandos, sondern durch den Aufruf automatisch generierter Funktionsbausteine.

Ein zentraler Sperrverwalter (Enqueue Service) hat die Aufgabe der Die Sperrverwaltung in verteilten Client/Server- und Multiprozessorsystemen. Der Sperrverwalter überwacht auch die Verträglichkeit der gewährten Sperren systemweit.

5.2.8 Programmierung grafischer Benutzerschnittstellen

Jede Dialoganwendung im System R/3 wird von Dynpros (Dynamisches Programm) gesteuert. Ein Dynpro ist eine Kombination einer Eingabemaske mit Programmcode (Module). Die Maske enthält alle grafischen Elemente des Eingabefensters, wie z.B. Eingabefelder, Check Boxen und Radio Buttons. Das Layout einer Maske wird mit dem *Screen Painter* (s. Abschnitt 5.3.1) festgelegt. Die Funktionalität des Dynpros wird durch Module festgelegt.

Über die Ablauflogik erfolgt eine Verbindung von Dynpros und Modulen. Die Ablauflogik ist ein kurzer Quelltext, in dem Module aufgerufen werden. Einige Module werden zur Initialisierung oder dynamischen Modifikation des Dynpros ausgeführt, bevor das Dynpro auf dem Bildschirm angezeigt wird (*Process before output*, PBO). Eine zweite Modulgruppe wird nach der Beendigung der Dateneingabe abgearbeitet (*Process after input*, PAI).

In einem Dynpro bestehen folgende Verzweigungsmöglichkeiten:

- Fortsetzung mit anderem Dynpro oder Dialogbaustein,
- Ausführung eines Reports und
- Start einer anderen Transaktion.

Zum Aufruf anderer Elemente existieren zwei Kommandos LEAVE und CALL mit einigen Varianten. Während LEAVE einen endgültigen Sprung zum anderen Element ohne automatische Rückkehr zur Aufrufstelle ausführt, kann aus dem mit CALL aufgerufenen Elementen mittels spezieller Return-Anweisung zur Aufrufstelle zurückgesprungen werden.

Dynprotypen

Man unterscheidet fünf verschiedene Dynprotypen:

- Normal: Die Dynpros nehmen bei Ausführungen die gesamte Arbeitsfläche ein. Sie verfügen über eine Ablauflogik und werden nacheinander abgearbeitet.
- Selektionsdynpro: Sie werden vom System generiert, wenn ein Report mit Parametern oder Selektionen zum erstenmal über eine Menüfunktion generiert wird. Die Selektionsdynpros besetzen eine nicht veränderbare Ablauflogik.
- Modales Dialogfenster: Spezielle Anweisungen der Ablauflogik erlauben es, ein beliebiges Dynpro als Unterprogramm aufzurufen. Wird das aufgerufene Dynpro beendet, erfolgt ein Rücksprung zum aufrufenden Dynpro. Diese Dynpros können auch als Popup angezeigt werden.
- Selektionsdynpro als modales Dialogfenster: Diese Dynpros werden ebenfalls generiert und sollten nicht manuell bearbeitet werden.
- Subscreen: Sie sind nicht eigenständig lauffähig und werden zur Laufzeit dynamisch in ein anderes Dynpro eingebunden. Auf derartigen Subscreens können Spezielle Informationen dargestellt werden.

Automatische Prüfung der Bildschirmfelder

Die Felder einer Bildschirmmaske beziehen sich auf Felder von Tabellen im ABAP/4-Dictionary. Datentyp und Länge des Bildschirmfeldes werden aus dem Dictionary übernommen. Dieses führt dazu, daß die folgenden Funktionen für Feldeigenschaften im Dynpro automatisch aus dem Dictionary übernommen werden:

- Überprüfung der Gültigkeit der Bildschirmeingaben gemäß dem definierten Wertebereich,
- Konvertierung zwischen interner und externer Darstellung der Ein- oder Ausgabe von Feldinhalten,
- Bereitstellung der Felddokumentation über Online-Hilfe.

Die grafische Aufbereitung der R/3-Standard-Bedieneroberfläche erfolgt durch das Programm SAP GUI, das auf jedem R/3-Frontend installiert ist. Zusätzlich ist SAP GUI für die Kommunikation zwischen Präsentation und Anwendungserver zuständig.

Die Mehrsprachigkeit des SAP-GUI

Die Benutzeroberfläche des Systems R/3 steht in 15 unterschiedlichen Landessprachen zur Verfügung. Die Unterstützung mehrerer Sprachen durch das SAP-GUI wird dadurch erreicht, daß alle textuellen Anteile in den Definitionen von Eingabeformularen und Ausgabelisten separat verwaltet werden. Sie werden erst zum Zeitpunkt der Anzeige in Abhängigkeit vom Sprachkennzeichen in das Bildschirmlayout integriert.

5.3 Entwicklungswerkzeuge

Die ABAP/4 *Development Workbench (DW)* ist eine integrierte Entwicklungsumgebung zur Erstellung und Bearbeitung von R/3- Anwendungen. Diese Workbench enthält alle Werkzeuge der Entwicklungsumgebung (Screen painter, Menu painter, ABAP/4- Compiler,etc.), um neue Anwendungen zu entwickeln. Alle Entwicklungsobjekte, insbesondere ABAP/4-Code, werden im Repository verwaltet.

In diesem Abschnitt werden die Besonderheiten dieser Werkzeuge vorgestellt.

5.3.1 Werkzeuge zur Implementierung

Die ABAP/4-Development-Workbench enthält die für die Erstellung und Pflege von ABAP/4-Programmen erforderlichen Werkzeuge. Neben dem Programmeditor stehen Dialogeditor und Menüeditor zum Erzeugen und Bearbeiten von Dynpros und grafischen Softwaremenüs zur Verfügung. Die Fehlersuche in ABAP/4-Programmen wird durch einen Debugger unterstützt, der eine Fehlersuche auch über Systemgrenzen hinweg ermöglicht.

Programm-Editor

Der Editor der ABAP/4-DW enthält neben den üblichen Editierfunktionen, wie z.B. Kopieren, Ausschneiden usw., auch einen *Pretty Printer* zur Strukturierung von Quellcode, eine Funktion zur Syntaxüberprüfung und viele andere Sonderfunktionen, welche den Aufwand für Programmentwicklung und Programmwartung reduziert. Besonders hervorzuheben ist hierbei die systemweite Navigationshilfe für den Programmierer.

Entwicklungswerkzeuge für grafische Benutzerschnittstellen

Der *Screen Painter* der ABAP/4-Development-Workbench hat die Aufgabe Dynpros zu erzeugen, zu bearbeiten, anzuzeigen und zu löschen. Zusätzlich können allgemeine Eigenschaften der Dynpros, wie z.B. die verwendete Landessprache oder die Nummer des Folge-Dynpros bestimmt werden.

Zur Bildschirmgestaltung können mit Hilfe des Grafik-Editors grafische Bedienlemente (z.B. *Push Buttons, Radio Buttons, Check Boxes,* usw.) in die Bildschirmmasken eingebunden werden. Parallel zum grafischen Editor kann sich der Entwickler eine beliebige Dictionary-Tabelle anzeigen lassen, um Felder aus dieser Tabelle in den Arbeitsbereich eines Fensters zu übernehmen. Die Übernahme mehrerer Tabellenfelder ist mit *Cut & Paste* möglich. Änderungen von Feldbezeichnungen und Ein-/Ausgabedefinitionen, die im ABAP/4- Dictionary hinterlegt sind, führen zu einer automatischen Aktualisierung des Dynpros.

Menu Painter

Die Aufgabe der *Menu Painter* ist das Zuordnen bestimmter Menüs, Funktions- und Drucktasten zu den von Programmieren realisierten Funktionen. Der Aufbau der Menüs und Drucktasten einer Anwendung wird auf einer zentralen Arbeitsoberfläche dargestellt. Einzelne Funktionen in den erstellten Menüs sind aktivierbar und deaktivierbar. Die Sprache eines Menüs wird beim Anlegen eines Programms vergeben. Um die Bedienung aller mit der ABAP/4-Workbench erstellten Oberflächen einheitliche zu darstellen, werden die Menüs und Drucktasten gemäß SAP-Style-Guide gestaltet.

Debugger

Der Debugger dient zur Fehlersuche in ABAP/4-Programmen und in Dynpros und bietet eine Einzelschrittfunktion für die Programmausführung. Der Debugger kann auf unterschiedliche Weise aktiviert werden. Generell besteht die Möglichkeit, Programme explizit im Debugging-Modus zu starten. Alternativ kann nach Programmabbrüchen durch Laufzeitfehler oder aus der Prozeßüberwachung des Computing-Center-Management-Systems heraus in den Debugger eingestiegen werden. Das Besondere an diesem Debugger ist die Einbeziehung von Remote Function Call (RFC), welche systemweit integriert werden (*Remote Debugging*).

Workgroup-Funktionalität

Für die Abwicklung von Software-Entwicklungsprojekten im Team stehen in der ABAP/4-Workbench der ABAP/4-Development-Organizer, das Transportsystem und ein Informationssystem zur Verfügung.

Workbench-Organizer

Der Workbench-Organizer umfaßt die Entwicklungsorganisation und -koordination. Er unterstützt sowohl kleine und große als auch zentrale und dezentrale Softwareentwicklung. Die Arbeitsweise eines Workbench-Organizers kann wie folgt zusammengefaßt werden:

- Es existieren Sicherheit- und Zugriffskontrollmechanismen für alle Entwicklungsobjekte. Die Entwicklung oder Pflege der Entwicklungsobjekte ist bis zur Freigabe des Auftrages für andere Entwickler gesperrt.

- Jedem Entwicklungsobjekt wird der verantwortliche Entwickler und eine Entwicklungsklasse zugeordnet, die das Sachgebiet des Objekts kennzeichnet. Dadurch lassen sich sofort Ansprechpartner zu jedem Objekt finden.

- Beim Einsatz mehrerer Entwicklungssysteme können die Entwickler ihre Objekte in ihrem lokalen Entwicklungssystem nach einem Check-Out isoliert entwickeln und testen. Zur Integration der Objekte aus mehreren lokalen Systemen müssen diese in ein gemeinsames Integrationssystem transportiert und getestet werden.

Das Transportsystem

Der Transport von Entwicklungsobjekten und Customizing-Einstellungen ist die Aufgabe eines Transportsystems. Für jeden Transport wird automatisch ein Transportprotokoll angelegt. Sollte sich z.B. ein Produktivsystem nach einem Import aus dem Testsystem fehlerhaft verhalten, so kann sofort festgestellt werden, welche Objekte transportiert wurden oder aus welchem Grund der Transport durchgeführt wurde.

Das Informationssystem

Ergänzend zum ABAP/4 Development-Organizer und Transportsystem steht ein Informationssystem für das Suchen, Anzeigen, Bearbeiten und Analysieren von Entwicklungsobjekten und Transporten zur Verfügung stehten. Beispielsweise können Entwicklungsobjekte selektiert und sortiert nach Sachgebieten, verantwortlichen Entwicklern, Änderungsdatum usw. angezeigt werden.

Repository

Alle ABAP/4-Entwicklungsobjekten, z.B. ABAP/4-Programme, Bildschirmbeschreibungen, Dokumentation und Hilfetexte, ABAP/4-Dictionary-Daten sind im ABAP/4 Repository gespeichert. Zur Auswertung der im ABAP/4 Repository enthaltenen Informationen steht das Repository-Informationssystem zur Verfügung. Es umfaßt die folgende Auswertungen:

- Auflistung von Programmen, Tabellen, Feldern, Datenelementen und Domänen. Es können bestimmte Optionen wie das Datum der letzten Änderung berücksichtigt werden.

- Verwendungsnachweis von Tabellen und ihren Feldern in Dynpros und ABAP/4-Programmen.

- Darstellung von Fremdschlüsselbeziehungen.

5.3.2 Software-Management-Werkzeuge

Der Einsatz von Software in verteilten Umgebungen wird im System R/3 durch folgende Werkzeuge unterstützt:

- **Automatische Software-Verteilung:** Die gesamte R/3-Anwendungs-Software wird nur einmal in der zentralen Datenbank installiert. Sie steht dann allen Anwendungs-Servern zur Verfügung. Eine Versionsüberwachung überprüft beim Aufruf einer Transaktion, ob die betroffenen Anwendungsmodule mit den aktuellen Modulen überein-

stimmen. Ist es nicht der Fall ist, wird automatisch und für den Anwender transparent ein Abgleich durchgeführt. Für die SAP-GUI-Software gibt es ein automatisches Verteilungsverfahren. Es sorgt dafür, daß neue Software-Versionen beim Start des Desktop-Systems automatisch vom Datenbank-Server geladen werden.

- **Systemmanagment:** Das Computing-Center-Management-System (CCMS) des Systems R/3 stellt unterschiedliche Funktionen innerhalb eines grafikorientierten Arbeitsumfeldes für den zuverlässigen Betrieb des Systems zur Verfügung. Diese Funktionen lassen sich wie folgt klassifizieren:
  - Systemüberwachung: Die Systemüberwachung sammelt relevante Leistungsdaten der R/3-Installation und stellt diese alphanumerisch dar.
  - Systemsteuerung: Die Komponenten der Systemsteuerung regeln die aktiven Abläufe innerhalb von R/3-Installationen, z.B. das Starten oder Herunterfahren von Systemen und Prozessen, Balancierung der Anwendungslast auf unterschiedlichen Rechnern, Einplanung und Überwachung von Datensicherungsaufträgen (Backup und Recovery).
  - Management-Schnittstellen: Zur Anbindung zusätzlicher Systemmanagementdienste bietet CCMS offene Schnittstellen, z.B. die Datenkollektor-Schnittstelle für die Kommunikationsnetze Ethernet, Token Ring und FDDI empfängt Netzwerkinformationen wie die aktuelle Segmentbelastung oder die Kollisionsrate.

Kapitel 6

Tycoon: Eine Entwicklungsumgebung für persistente Objektsysteme

Das Tycoon-System ist eine moderne persistente Programmierumgebung zur Entwicklung persistenter Objektsysteme. Die Zielsetzung bei der Entwicklung von Tycoon umfaßt die Gestaltung angemessener sprachlicher und architektonischer Grundlagen zur Definition und Integration generischer Dienste in offenen Systemumgebungen [Matthes 93]. Im Tycoon-System lassen sich folgende drei Komponenten als wesentliche Bestandteile identifizieren:

- eine schlanke Systemarchitektur (die Dienste traditioneller Programmiersprachen, Datenbanksysteme und Betriebssysteme entkoppelt und auf geeignete Weise so zur Verfügung stellt, daß die Entwicklung skalierbarer Anwendungssysteme sowohl für einfache PC-Umgebungen als auch für vernetzte Mehrbenutzer- und Client/Server-Architekturen ermöglicht wird.)
- eine auf einem kleinen Kern reduzierte persistente Programmiersprache höherer Ordnung,
- ein Framework bestehend aus erweiterbaren Bibliotheken für die Unterstützung von Anwendungsentwicklern (Dieser Framework bietet vordefinierte, wiederverwendbare Dienste zur Behandlung von Massendaten und Schnittstellen für kommerziellen Diensten wie z.B. StarView, DCE-RPC, SUN-RPC, usw.)

Die Beschreibung dieser Komponenten ist Gegenstand dieses Kapitel. Die wohldefinierten Schnittstellen der Systemarchitektur, die die Offenheit des Systems gewährleisten, werden im nächsten Abschnitt erläutert. Im Abschnitt 5.2 werden Konzepte der Programmiersprache in die Referenzarchitektur (s. Kapitel 3) eingeordnet und ihre speziellen Aspekte betrachtet. Im Vergleich zu kommerziellen Produkten verfügt Tycoon über einen begrenzten Satz von Werkzeugen, die im Abschnitt 5.3 beschrieben werden.

6.1 Die Architektur des Tycoon-Systems

Die Architektur des Tycoon-Systems ist durch ein Schichtenmodell gekennzeichnet (Abbildung 6.1).

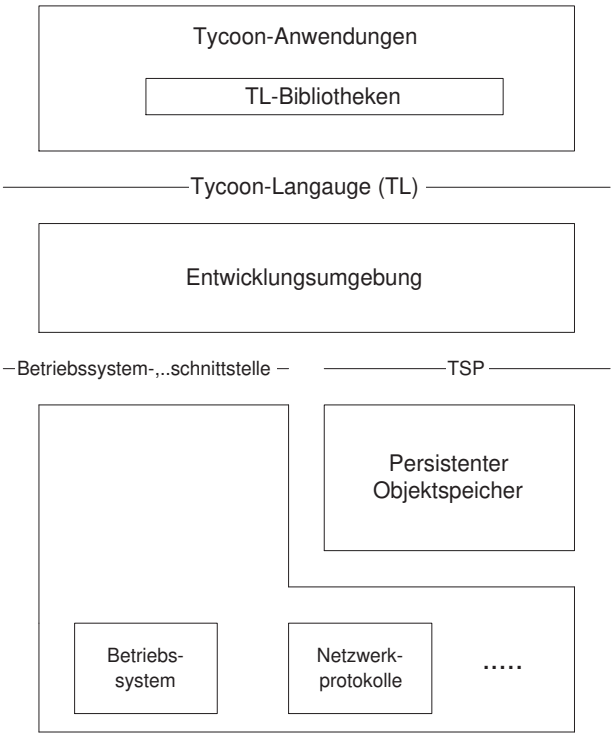

Abbildung 6.1: Die Schichtenarchitektur des Tycoon-Systems

Klar abgegrenzte konzeptionelle Schichten mit wohldefinierten Schnittstellen spiegeln die strikte Trennung von Aufgaben und Methoden der Repräsentation, der Datenmodellierung, Datenmanipulation und Datenspeicherung wieder. Diese Trennung unterstützt die Skalierbarkeit und Offenheit der Systemumgebung.

6.1.1 Betriebssystemschnittstelle

Die Architektur des Tycoon-Laufzeitsystems (RuntTime (RT), Tycoon Machine (TM), Tycoon Language (TL)) ist stark von den Gesichtspunkten der Portabilität geprägt. RT schirmt den gesamten Rest des Laufzeitsystems von Betriebssystemspezifika ab. Abhängig von der zugrundeliegenden Hardware und des Betriebssystems ist lediglich die Tycoon-Maschine. Sie stellt in plattformunabhängiger Weise Funktionen für Textein- und Ausgabe, Dateizugriffe und Speicherverwaltung zur Verfügung [Mathiske 96]. Der Anpassungsaufwand bei einer Portierung des Systems beschränkt sich also auf Funktionen in diesem Laufzeitsystem.

6.1.2 Netzwerkschnittstelle

Aufgrund der dominierenden internationalen Verbreitung von TCP/IP wird dieses Netzwerkprotokoll in Tycoon unterstützt. Die Kommunikationsprogrammierung mit TCP/IP findet über eine plattformunabhängige schmale Socket-Schnitstelle [Mathiske 96] statt. Damit ist eine größt mögliche Plattformunabhängigkeit des Tycoon-Objektsystems bezüglich der Netzwerkkommunikation gewährleistet.

6.1.3 Datenbanksystemschnittstelle

Die voll integrierte persistente Programmierumgebung Tycoon benutzt die abstrakte Objektspeicherschnittstelle TSP (Tycoon Store Protocol) als Speicherprotokoll zur Speicherung und Manipulation persistenter Daten. Die TSP Schnittstelle ist datenmodellunabhängig und abstrahiert von der Implementierung spezieller Objektspeicher. Ein TSP-Benutzer kann zwischen verschiedenen Speichern wählen und mit mehr als einem persistenten Speicher gleichzeitig arbeiten. Diese Schnittstelle ermöglicht die folgende Interaktionen zwischen TL-Client-Programm und Objektspeicher-Server:

- Uniformer Zugriff auf persistente und nicht-persistente Objekte, polymorphe Datenstrukturen, Funktionen höherer Ordnung und
- dynamische Schemaänderungen und dynamische Bindungstechniken.

Eine detaillierte Beschreibung der TSP-Schnittstelle befindet sich in [Matthes et al. 95].

Darüber hinaus integriert eine strikt typisierte generische SQL-Schnittstelle die Funktionalität einer SQL-Datenbank in das Tycoon- Programmierszenario.

6.1.4 Schnittstellen zu externen Anwendungen

Das Tycoon-System erlaubt die Integration der externen Anwendungsdienste in die typsichere, polymorphe Sprache TL. Zur Definition neuer und zur Einbindung bestehender externer Dienste wird die polymorphe Programmiersprache höherer Ordnung TL verwendet. In Integrationsbibliotheken werden die externe Anwendungsdienste über Gateway-Konstruktionen auf Objekte der Programmiersprache TL abgebildet. Auf dieser Weise integrierte Dienste profitieren automatisch von den Eigenschaften des Tycoon-Systems wie Persistenz, Mobilität und Typsicherheit.

Bereits realisierte Anwendungs- und Systemdienste sind:

- Tycoon File System (TFS): Das TFS ist eine Dateischnittstelle zur persistenten Objektspeichern unter Verwendung des Network-File-System und des Tycoon-Store-Protocols [Gorzynski 95]. Es erlaubt UNIX-Anwendungen, wie z.B. emacs Editoren, WWW-Browser, etc., den entfernten Zugriff auf die in Objektspeichern gespeicherten Daten und ihre Manipulation mit Hilfe von Standard UNIX-Befehlen. Die Dateisystemobjekte von UNIX-Anwendungen können ebenfalls von Tycoon-Programmen verwendet werden.
- INQUERY: In [Steffens 96] wird die Anbindung des INQUERY an Tycoon beschrieben. INQUERY ist ein Information Retrieval System. Es erlaubt die rechneradäquate Repräsentation von Textsammlungen und die Auswertung von Anfragen zu deren Inhalten. Eine Schnittstelle im Tycoon-System ermöglicht den Zugang zur INQUERY-Datenbanken, das Parsen von Anfragen, deren Auswertung und den Zugang zur Dokumenten und Informationen der Textsammlung. Die Orthogonale Persistenz im Tycoon-System wird benutzt zur langfristigen Speicherung von Anfrageergebnissen und zum Anlegen von Standardanfragen.

6.1.5 Schnittstelle zu anderen Sprachen

Wie es in Abbidung 6.2 dargestellt ist, kann der TL-Anwendungscode von Hauptprogrammen aufgerufen werden, die in anderen Sprachen geschrieben sind. Es kann das gesamte

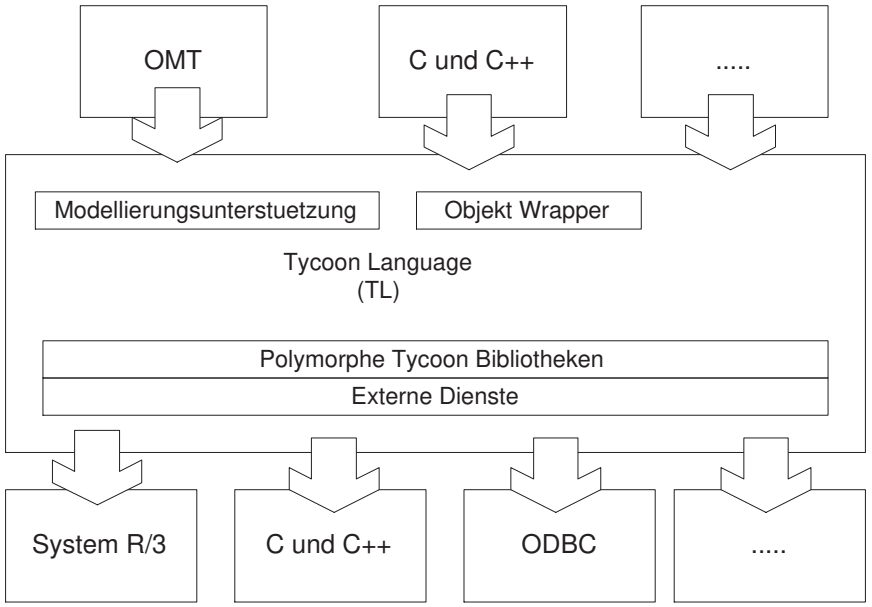

Abbildung 6.2: Das offene persistente Objektsystem Tycoon

Tycoon-System als Subsystem anderer Programme fungieren. Andererseits werden bestehende und neu entwickelte externe Bibliotheken in uniformer Weise als TL-Bibliotheken in das Tycoon-System integriert. Dadurch ist ein typsicherer Zugriff auf externe Daten und Code sichergestellt.

Integrierte externe Funktionen werden syntaktisch nicht von TL implementierten Funktionen unterschieden. Es existieren Bereits folgende Schnittstellen zu anderen Sprachen:

- Schnittstelle zu C und C++: Tycoon bietet eine bidirektionale Schnittstelle zwischen TL und C, die eine gegenseitige Integration der Sprachparadigmen ermöglicht [Mathiske 92]. Externe C-Funktionen (z.B. aus C-Bibliotheken) können in TL wie gewöhnliche TL-Werte eingebunden werden. Funktionen in C sind keine gleichberechtigten Werte, es ist jedoch möglich, einen Zeiger auf eine Funktion zu definieren. Die TL-Funktionen werden so gekapselt, daß sie als Zeiger an C-Programmen übergeben werden können. Damit sind sie mit ihrer vollen Funktionalität in C-Anwendungen integriert.

  In [Geisler 95] ist auf Basis dieser C-Schnittstelle ein C++ Gateway entwickelt worden. Die Architektur des Gateways umfaßt die Abbildung programmiersprachlicher Konzepte aus C++, wie Typen, Klassen, Vererbung, Overriding und Callbacks, auf entsprechende Konzepte in TL (Typen, Typoperatoren, Module). Dieses Gateway ermöglicht z.B. die Anbindung der portablen C++-Klassenbibliothek Starview zur Integration grafischer Dienste in die persistene Tycoon-Umgebung.

- Die bidirektionale Schnittstelle zum System R/3: Durch ein bidirektionales Gateway zwischen Tycoon und System R/3 ist die Bibliothek RFC (*Remote Function Call*), welche die Grundlage verteilter Client/Server-Programmierung des Systems R/3 bildet, in die Tycoon-Programmierumgebung integriert worden [Lutz 97]. Aus TL können Funktionsbausteine des Systems R/3 und aus SAP R/3 TL-Funktionen aufgerufen werden. Außerdem wird der mehrfache Rückruf unterstützt.

Diese Schnittstelle erlaubt die typsichere Integration von R/3-Funktionen in das Tycoon-System. Alle Parameter werden dynamisch typgeprüft.

6.1.6 Fenstersystemschnittstelle

Die Bibliotheken, StarView und NeWS-Toolkit [1] schirmen die Tycoon-Anwendungen von darunterliegenden Fenstersystemen ab. Die systemübergreifende Klassenbibliothek StarView stellt eine Mischform der Schnittmengen und des Obermengenansatzs dar und ermöglicht die einheitliche Darstellung grafischer Benutzeroberflächen auf OSF/Motif, OpenWindows, Finder und Windows. NeWS Toolkit ist eine Objektorientierte Programmierschnittstelle zum Fenstersystem OpenWindows mit Stilrichtlinien OPEN LOOK.

6.2 Das Tycoon-Programmiermodell

Die einheitliche sprachliche Basis für alle Programmieraufgaben in Tycoon stellt die Tycoon-Programmiersprache (TL) dar. Neben der Anwendungsprogrammierung unterstützt TL auch die System-, Werkzeug und ad hoc-Programmierung [Matthes 93]. Gemeinsames Ziel ist dabei ein hohes Maß an Typsicherheit, sowohl bei der Anwendungsentwicklung als auch bei der Erstellung generischer Systemdienste.

Die Sprache ist nicht auf die Verwendung eines bestimmten Datenmodells festgelegt. Insbesondere können funktionale, relationale und objektorientierte Modellierungsansätze unterstützt werden. Für die Implementierung von TL gelten die gleichen Forderungen wie für die Tycoon-Architektur: Portabilität und Skalierbarkeit sollen die Einsatzmöglichkeit auf unterschiedlichen Hard- und Softwareplattformen gewährleisten. Zusammengefaßt ergibt sich folgende Charakteristierung der Sprache TL. TL ist algorithmisch vollständig, funktional, imprativ, strikt evaluativ, deterministisch, persistent, strikt typisiert, explizit typisiert, polymorph und modular.

Im nächsten Abschnitt sind die für die Anwendungsentwicklung relevante Konzepte der Sprache beschrieben. Diese Beschreibung orientiert sich im wesentlichen an [Matthes 93] und [Matthes et al. 94].

6.2.1 Datenstrukturierungskonzepte

In TL wird ein einheitliches Konzept für die Behandlung sowohl vordefinierter als auch benutzerdefinierter Datentypen verwendet. Durch diese Homogenisierung werden folgende Ziele erreicht:

- Skalierbarkeit: Die vordefinierten semantischen Objekte können neu definiert werden, z. B. eine Exakte Finanzarithmetik oder längere Zeichenkettenrepräsentationen.
- Portabilität: Durch Lokalisierung potentiell systemabhängiger Bibliotheksfunktionalität wird eine höherer Portabilität von TL- Anwendungen erreicht.

Vordefinierte Datentypen

Aus Gründen der Homogenisierung zwischen vordefinierten und benutzerdefinierten Datentypen sind in TL im Gegensatz zu vielen traditionellen Programmiersprachen weder Basistypen, noch Konstanten der Basistypen oder Funktionen auf den Basistypen im Typsystem vordefiniert. Statt dessen sind diese in den Tycoon Standardbibliotheken definiert.

[1] Toolkits sind Bibliotheken, die von den Bibliotheken der Fenstersysteme abstrahieren [Müßig 94].

So müssen die Namen der Basistypen (z.B. bool.T), ihre Konstanten (bool.true) und die auf den Basistyp definierten Funktionen (bool.and) explizit aus Modulen dieser Bibliotheken importiert werden. Einzige Ausname bilden die Literale der Basistypen int, char, string und real. Sie sind im Compiler fest eingebaut und können wie vordefinierten Basistypen und Funktionen angesprochen werden. Eine implizite Wertumwandlung zwischen den Basistypen ist nicht vorhanden.

Benutzerdefinierter Datentypen

Auf der Ebene der Typen erlaubt TL ausgehend von Typparameteren die Konstruktion neuer benutzerdefinierter Typen. Die Typparameter können bei der Anwendung durch verschiedene Ausdrücke instantiert werden. Funktionen, die Typen auf Typen abbilden, werden in TL durch Typoperatoren eingeführt. Neue Datenstrukturen werden durch die Anwendung von Typoperatoren auf applikationsspezifische Datentypen erzeugt. Die Konstruktion abstrakter Datentypen (ADT) läßt sich unmittelbar mit den in TL vorhandenen Sprachkonzepten realisieren. Die Aggregation von Datentypen und Operationen läßt sich in TL anhand eines Tupels nachbilden.

Typkonstruktoren

TL bietet einen orthogonalen Satz primitiver Konstruktoren. Basierend auf der geschachtelten und rekursiven Anwendung dieser Primitive finden sich in den Tycoon- Standardbibliotheken (benutzerdefinierte) Typkonstruktoren zur Modellierung von Mengen, Listen, Relationen, Klassen, Dateien, etc., und polymorphen Funktionen zur Manipulation von Werten dieser parametrisierten Typen. Hier werden die vordefinierten Typkonstruktoren Tupel, Tupel mit Varianten und Records vorgestellt.

- **Tupel** realisieren einen kartesischen Produkttyp. Tupelwerte sind geordnete Folgen von Bindungen. Der Sichtbarkeitsbereich der Feldnamen beschränkt sich auf die Tupel-Konstruktionen, wobei mittels Punktnotation Felder referenziert werden können. Als Tupelkomponenten können auch Funktionen benutzt werden. Eine wichtige Eigenschaft des Typsystems von TL ist die Unterstützung der Subtypisierung (s. Abschnitt..). Die Subtypbeziehung ist auch für Tupeltypen definiert. Ein Tupeltyp A ist eine Subtyp eines Tupeltyps B, wenn die Komponenten von A in der gleichen Reihenfolge mit den Komponenten von B übereinstimmen und eventuell weitere Komponenten am Ende hinzugefügt werden.

- **Tupel mit Varianten** erlauben die Modellierung von Alternativen. Jede Variante wird durch einen Namen gekennzeichnet und enthält eigene Signaturen. Der Zugriff auf Varianten erfolgt durch eine Fallunterscheidung.

- **Records**: Neben der Möglichkeit Werte durch Tupelkonstruktoren zusammenzufassen, gibt es in TL noch eine weitere Form der Aggregation. Ein Recordtyp definiert eine ungeordnete Menge von Bindungen. Die Sichtbarkeit der Feldnamen sind auf den Recordtyp bzw. den Recordkonstruktor beschränkt.

  Der Zugriff auf Record-Komponenten erfolgt wiederum über die Punktnotation. Die Subtypbeziehung zwischen Recordtypen erfolgt ähnlich wie bei Tupeltypen. Neben der Spezialisierung auf die Komponentenebene ist hier das Hinzufügen neuer Felder an beliebiger Stelle möglich, weil die Signaturen in einem Recordtyp ungeordnet sind.

Darüber hinaus erlaubt TL die inkrementelle Erweiterung existierender Recordwerte um zusätzliche Komponenten.

Das Typsystem

Das Typsystem von Tycoon garantiert die vollständige Typkorrektheit der entwickelten Datenbanksoftware und ermöglicht die Definition generischer Systemdienste. Realisiert werden diese moderne Eigenschaften durch die Unterstützung folgender Konzepte in TL:

- strikte Typisierung: In TL werden durch Typregeln formalisierte Konsistenzbedingungen auf Termen durch eine statische Programmanalyse geprüft. Durch explizite Sprachkonzepte, z.B. dynamische Typen, kann dieser Konsistenztest kontrolliert auf wohldefinierte Zeitpunkte während der Programmlaufzeit verzögert werden.
- strukturelle Äquivalenz: Subtypisierung in TL beruht auf struktureller Kompatibilität.
- Dynamische Typisierung: Zur Abdeckung der Anwendungsgebiete, in denen die Typüberprüfung nicht vollständig zur Übersetzungszeit möglich ist und dynamisch zur Laufzeit erfolgen muß, erlaubt TL die Definition von dynamischen Typvariablen.
- Polymporhismus: Das Typsystem von TL ist polymorph. Damit können in TL generische Algorithmen mit typunabhängigem Verhalten realisiert werden. In TL sind fast alle gängigen Polymorphiekonzepte vorhanden, die heutzutage im Rahmen eines *statisch typsicheren* Systems unterstützt werden können:
  - Parametrische, polymorphe Funktionen in TL sind Funktionen, die Typvariablen als Parameter besitzen. Diese Typvariablen können bei der Funktionsanwendung instanziiert werden.
  - Subtyppolymorphismus: TL verfügt über Regeln zur Definition von Subtypbeziehungen auf Typen. Die Subtypbeziehungen werden ausgehend von allgemeinen Supertyp **OK** induktiv definiert. Damit sind die vordefinierten Typen Bool, Int und String alle Tupeltypen, Recordtypen und Funktionstypen Subtyp von **Ok**.

    Die Subtypbeziehung ist auch auf komplexen Typen definiert. Ein Tupeltyp S ist ein Subtyp eines Tupeltyps T, falls die Folge von Signaturen in T ein Präfix der Signaturen von S ist. Die Subtypbeziehung zwischen Recordtypen erfolgt ähnlich wie bei Tupeltypen. Da die Signaturen in einem Recordtyp ungeordnet sind, kann die Spezialisierung durch das Hinzufügen neuer Felder an beliebiger Stelle erfolgen. Die Regeln für die Subtypbeziehung zwischen Funktionstypen gehen dabei auf die Kontravarianzregel zurück [Matthes 93].

6.2.2 Strukturierungskonzepte für die Programmierung im Kleinen

Die Konzeption der Sprache TL ist nicht auf einen besonderen Programmierstil fixiert. Neben den funktionalen Sprachkonzepten (Funktionsabstraktion, bedingte Evaluation) sind in TL weitere Sprachprimitiven vorgesehen, die den Sprachkern um die Basismechanismen der imprativen, objekt-orientierten und modularen Programmierung erweitern. In diesem Abschnitt werden die imprative Sprachelemente vorgestellt, die neben veränderlichen Variablen und destruktiven Zuweisungen auch eine Änderung des Kontrollflusses durch bedingte Ausdrücke, Schleifen und Ausnahmen beinhalten.

Blockstrukturierte Kontrollstrukturen

Der Ablauf von TL-Programmen ist durch die sequentielle Auswertung von Ausdrücken bestimmt. Mehrere Ausdrücke können in **begin end**-Blöcken zusammen gefaßt werden. Neben Sequenzen gibt es Kontrollstrukturen zur bedingten Ausführungen (**if then**) Fallunterscheidungen (**case when else**) und drei Arten von Schleifen (**loop, while, for**).

Prozedurale Abstraktion

Eine sehr moderne und vollständige Form der Abstraktion sind Funktionen in Tycoon. Die einfachste Form einer Funktion besteht aus dem Schlüsselwort **fun**, gefolgt von einer Liste von Formalparametern und einem Funktionsrumpf. Die Parameterliste einer Funktion wird durch eine Folge von Signaturen ausgedrückt, aber kann auch leer sein. Den Funktionsrumpf bildet ein Wertausdruck, dessen Resultat bei Funktionsanwendung dem Ergebnis der Funktion entspricht. Wird eine Funktion als eine ergebnislose Prozedur nur zur Erzeugung von Seiteneffekten benötigt, so wird als Resultat vom Programmierer der Wert **ok** zurückgeliefert.

In einem Funktionsrumpf sind die global zum statischen Sichtbarkeitsbereicht definierten Namen, die im Rumpf lokal gebundenen Namen und die durch die Formalparameter angegebenen Namen sichtbar. Funktionen lassen sich an Wertvariablen binden. In TL sind Funktionen gleichberechtigte Werte (*first class values*). Sie können daher insbesondere auch als Funktionsargumente und Funktionsergebnisse auftreten. Funktionen mit diesen Eigenschaften werden als Funktionen höherer Ordnung bezeichnet.

Neben Funktionen höhere Ordnung bietet TL zusätzlich die Möglichkeit, rekursive Funktionen, polymorphe Funktionen und Funktionstypen zu definieren. Die Definition der Subtypbeziehung zwischen Funktionstypen erlaubt die Parametriesierung von Funktionen höherer Ordnung. Funktionen erlauben eine dynamische Bindung der Argumente zum Zeitpunkt der Funktionsanwendung.

Ausnahmenbehandlung

Das Verhalten von Programmen in Fehlersituationen kann in TL durch den Mechanismus der Ausnahmebehandlung gesteuert werden. Beim Auftreten einer Ausnahme wird die Auswertung von Ausdrücken unterbrochen und ein Ausnahmepaket an die aufrufende Funktion weitergereicht. Wird die Ausnahme nicht vom Programm behandelt, so stoppt die Ausführung mit einer Fehlermeldung in der interaktiven Arbeitsumgebung. Die Prüfung auf Ausnahmesituationen erfolgt in Programmen mit Hilfe des **try**-Konstrukts. Jede Ausnahme enthält eine textuelle Beschreibung der Fehlersituation und kann in weiteren Bindungen zusätzliche Informationen zur Fehlerdiagnose liefern. Die Definition eigener Ausnahmen ermöglicht eine anwendungsspezifische Fehlerbehandlung. Ausgelöst werden Ausnahmen durch die **raise**-Anweisung.

Bei einer Anbindung externer Dienste sind Mechanismen zur Behandlung der extern ausgelösten Ausnahmen erforderlich. Bei der Anbindung von externen C-Bibliotheken und R/3-Funktionsbausteine sind Mechanismen zur Behandlung der extern ausgelösten Ausnahmen vorgesehen. Zum Beispiel werden bei einer Anbindung externer C-Bibliotheken Ausnahmesituationen automatisch auf TL-Ausnahmen abgebildet. Anwendungsverursachte Ausnahmen, die im aufgerufenen Funktionsbaustein ausgelöst werden, werden direkt in eine korrespondierende TL-Ausnahme übersetzt. Lutz [Lutz 97] gibt eine ausführliche Beschreibung der Behandlung von Ausnahmen, die bei der Verwendung der Tycoon-R/3

Schnittstelle ausgelöst werden.

Multi-Threading

Das Thread-Konzept ist als *add-on* Ansatz im Tycoon-System realisiert und orthogonal in das sprachliche Gefüge von TL integriert [Mathiske 96]. Im Tycoon-System können Anwendungsprogramme abgesehen von technischen Beschränkungen beliebig viele unabhängige Threads erzeugen, die auf eine geteilte Menge persistenter Daten, Module und wiederum Threads auf einem gemeinsamen Objektspeicher zugreifen. Damit können die Diensterbringer in Tycoon multithreaded gestaltet werden.

6.2.3 Strukturierungskonzepte für die Programmierung im Großen

Die Aufteilung von Programmen zwecks Realisierung großer Softwaresysteme wird in TL durch Schnittstellen und Module unterstützt. Diese stellen jeweils separate Kompilationseinheiten dar, wodurch auch die Entwicklungszeiten verkürzt werden. Die Gruppierung von Modulen und ihren Schnittstellen in Form von Bibliotheken trägt weiter zur Strukturierung bei.

Neben Modularisierungsmechnanismen ist die Wiederverwendbarkeit von Code eine Grundvoraussetzung für die Erstellung von Softwaresystemen. Die Wiederverwendung von Code wird in TL durch das moderne Konzept des parametrischen Polymorphismus und Subtyppolymorphismus unterstützt. Nachfolgend wird erläutert, wie die Entwicklung großer Softwaresysteme in TL durch separate Kompilationseinheiten und Polymorphismus unterstützt wird.

Modularisierungsmechansimen

Die Modularisierung ist neben der Funktionsabstraktion die wichtigste Strukturierungs möglichkeit moderner Programmiersprachen. TL-Programme setzen sich aus Modulen zusammen, die durch getrennte Schnittstellen (**interface**) und Implementations-Teile (**module**) beschrieben werden. In TL können zu einer Schnittstelle beliebig viele **module** existieren. In Schnittstellen werden die Signaturen für Funktionen, Typen und Typoperatoren definiert. Die Module können lokale Funktionen, Typen usw. besitzen, die nicht durch ihre Schnittstelle anderen Modulen zugänglich sind.

Ein Modul, das Funktionen anderer Module nutzen will, muß diese zuvor importieren. Es stehen dazu die Funktionen zur Verfügung, die in der Schnittstellen des betreffenden Moduls exportiert werden. Nach einem Import kann mit Hilfe der Punktnotation auf die Modulkomponenten, d.h. Funktionen und Typen zugegriffen werden.

Wiederverwendung von Code

Das Typsystem von TL bietet verschiedene Mechanismen, um die effiziente Entwicklung großer Softwaresysteme zu unterstützen. Ein wichtiges Mittel für die Wiederverwendbarkeit von Softwarebausteinen bietet das Konzept des Polymorphismus. Paramterische, polymorphe Funktionen in TL, die Typvariablen als Parameter besitzen, werden bei der Funktionsanwendung mit entsprechenden Datentyp instanziiert. Damit können Dienste realisiert werden, die auf eine Vielzahl von Argumenten anwendbar sind.

Subtyppolymorphismus erlaubt die Definition von Algorithmen, die für einen bestimmten Typ T entworfen sind, und unverändert auch mit Werten eines Untertyps von T funktionieren.

6.2.4 Verteilungsabstraktion

Im Rahmen der Arbeit von [Mathiske 96] wurde eine Erweiterung des persistenten Objektsystems Tycoon zu einer offenen Systemumgebung entworfen und implementiert. Durch Unterstützung einer möglichst direkten Abbildung verteilter Anwendungsstrukturen in korrespondierenden Programmiersprachenkonstrukte wird eine signifikante Erhöhung der Flexibilität und Qualität bei gleichzeitiger Minderung des Aufwandes der Entwicklung datenintensver verteilter Anwendungen erreicht [Mathiske 96]. Der Kern der Arbeit besteht in der Bereitstellung eines mobilitätsorientierten RPC-Mechanismus und in der Realisierung aktiv migrierender persistenter Threads als Konstrukte für autonome migrierende Aktivitäten (Workflows), die langlebig sind. Im folgenden Abschnitt werden diese Konzepte in Anlehnung an die Arbeit von [Mathiske 96] vorgestellt.

Entfernter Funktionsaufruf

Die Unterstützung für die Programmierung von einer Client/Server-Anwendung ist über einen *add-On* Ansatz in Tycoon integriert. Tycoons RPC-Mechanismus ist durch die besonderen Eigenschaften des Tycoon-Systems wie polymorphes Typsystem höherer Ordnung, dynamische Typvergleiche, Funktionen höherer Ordnung und orthogonale Persistenz geprägt. Dabei sind die folgende Eigenschaften von besonderer Bedeutung:

- Es gibt keine Beschränkung hinsichtlich der Typen der entfernten Funktionen. Insbesondere können entfernte Funktionen polymorph sein. Sie können rekursive Datenstrukturen, Funktionen und Threads als Argumente und Rückgabewerte besitzen. Die Übergabe von Parametern besitzen *call by value* Semantik.
- Server-Prozesse können im Netzwerk verlagert werden, ohne daß Client-Bindungen deshalb explizit aktualisiert werden müssen.
- Polymorphe persistente Client/Server-Programmierung in Tycoon erfordert den Austausch von Programmcode zwischen Client und Server. Diese kann zeitkritisch für die Netzwerkverbindung und platzkritisch für den Empfänger des Codes werden. Die automatische Replikation von Funktionen verhindert den wiederholten Austausch und die mehrfache Installierung des Codes auf der Empfängerseite.
- Für die Beschreibung von entfernten Tycoon-Diensten wird keine *Interface Definition Language* benötigt. Diese erfolgt wegen der überlegenen Eigenschaften auf der Typ- und Wert-Ebene in TL. Dadurch ist keine zusätzliche externe Beschreibung der Schnittstellen notwendig, wodurch bei der Wartung bestehender Systeme Fehlerquellen vermeiden werden.
- Eine Aufrufanforderung einer Funktion führt zu genau einem Funktionsaufruf.
- Die Dienstbringer in Tycoon sind multithreaded gestaltet.
- Die Bindung an entfernte Funktionen erfolgt dynamisch zur Laufzeit. Es ist weder erforderlich, Stub-Generatoren zu aktivieren noch Quelltexte zu compilieren oder den Anwendungscode neu zu linken.

Mobilität von Code

Der plattformunabhängige, mobilitätsorientierte RPC-Mechanismus in Tycoon ermöglicht die Mobilität von Objekten beliebigen Types in einer heterogenen Umgebung. Die semantische Objekte des Tycoon-Systems (Daten, Code und Threads), unterliegen einer orthogonalen Mobilität. Im Vergleich zur konventionellen Client/Server-Kommunikation verteilter Systeme, in der nur Datenwerte als Parameter eines entfernten Prozeduraufrufs definiert werden können, kann beliebiges Tycoon-Objekt als RPC-Argument oder Rückgabewert verwendbar sein. Besonders dabei sind die Tycoon-Objekte, die Code enthalten (Funktionens) bzw. Zustände und Code kapseln (Funktionsabschlüsse) [Mathiske 96]. Der entfernter Client kann sich damit auf gleicherweise sowohl Datensätze als auch Dienste verschaffen. Bei der Ausführung der mobilen Dienste auf der Client-Seite können weitere benötigte Tycoon-Objekte vom Server gefordert werden.

Migrierende Prozesse

Die Verteilungsabstraktion der Prozesse (Aktivitäten), die verteilte Aktivitäten wie Workflows realisieren, ist in Tycoon durch die direkte Abbildung der migrierenden Aktivitäten auf Threads ermöglicht. Persistente Threads sind ein Sprachkonzept zur Modellierung konkurenter Aktivitäten und bieten sich als Konstrukte für autonome migrierende Aktivitäten an, die langlebig sind. Um autonome Objekte wie mobile Agenten oder Workflows zu implementieren, können Threads ihre eigenen Migrationen selbst veranlassen. Darüber hinaus sind die Threads in der Lage, dynamische Bindungen an die jeweiligen lokalen Objekte einzugehen und wieder zu lösen, um lokale Ressourcen zu nutzen und mit anderen Aktivitäten zu kooperieren [Mathiske et al. 95].

6.2.5 Persistenzabstraktion

In Tycoon existiert auf sprachlicher Ebene keine Unterscheidung zwischen kurzlebigen und langlebigen Daten. Alle Werte, Funktionen und Typbindungen können persistent sein. Alle Bindungen und Modulimporte, die auf dem sogenannten *top level* der interaktiven Umgebung eingeführt werden, sind automatisch persistent. Ausgehend von diesen Bindungen wird die Persistenz weiterer Objekte über transitive Erreichbarkeit bestimmt. Nicht mehr benötigte Bindungen werden mittels Garbage Collection freigegeben.

6.2.6 Massendatenverarbeitung

Für die Erstellung datenintensiver Anwendungen in Tycoon wird ein *add-on* Ansatz verfolgt. Die Funktionalität zur Bearbeitung von Massendaten ist nicht fest in das System eingebaut, sondern wird von folgenden Bibliotheken zur Verfügung gestellt (Abbildung 6.3).

- Die Bibliothek *bulkenv* beinhaltet Typoperatoren zur Definition der Massendatentypen im Tycoon-Objektspeicher. Zur Manipulation dieser Daten bietet das Modul *iter* Funktionen zur Iterationsabstraktion. Diese Umgebung ist nicht an ein bestimmtes Datenmodell gebunden. Sie stellt die Dienste in modellunabhängiger Form zur Verfügung.
- Die Bibliothek *sql* [Kornacker 93] erlaubt Tycoon-Anwendungen die Integration von SQL-Datenbanken. Diese Schnittstelle ermöglicht die Definition und Manipulation relationaler Tabellen.

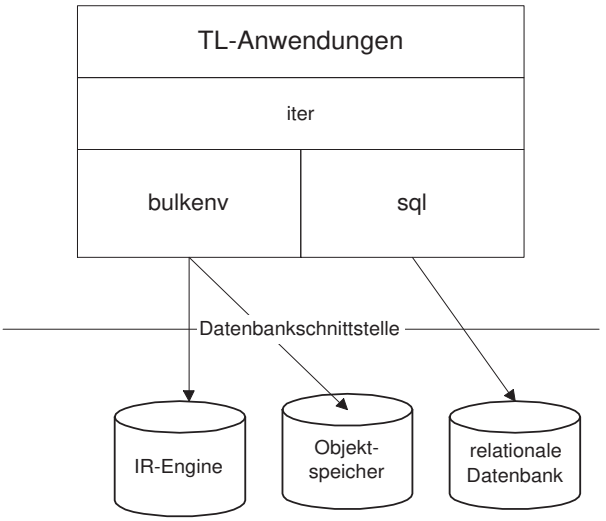

Abbildung 6.3: Massendatenverarbeitung in Tycoon

In diesem Abschnitt werden die Dienste dieser Bibliotheken zu Datendefinition und Datenmanipulation vorgestellt.

Sprachkonzepte zur Datenbankdefinition

Die Funktionalität zur Bearbeitung von Massendaten wird von den Bibliotheken *bulkenv* und *sql* zur Verfügung gestellt. Bei der Definition relationaler SQL-Tabellen ist eine explizite Umsetzung der Tycoon-Typinformationen in korrespondierende SQL-Attributdefinitionen notwendig. Zur Programmlaufzeit wird eine dynamische Bindung zwischen Tycoon-Programmvariablen und SQL-Tabellen durchgeführt. Erst zu diesem Zeitpunkt kann überprüft werden, ob die Attributbeschreibungen mit dem aktuellen SQL-Datenbankschema übereinstimmen.

Die Bibliothek *bulkenv* enthält viele Standarddatentypen zur Modellierung von Mengen, Listen, Kellerspeichern, Binärbäumen, usw. Durch ihre polymorphe Implementierung können sie Elemente beliebigen Typs aufnehmen. Weiterhin besteht die Möglichkeit, daß der Benutzer neue Bulk-Typen definiert, die seinen speziellen Anforderungen angepaßt sind [Niederee 92].

Sprachkonzepte zur Datenmanipulation

Eine wesentliche Eigenschaft vieler Datenbankprogrammiersprachen ist hinsichtlich des Zugriffs auf Daten die Abstraktion von konkreten Details ihrer Organisation. Kernpunkt dieser Abstraktionen in Tycoon ist der parametrisierte abstrakte Datentyp Iterator, der den lesenden Zugriff auf homogene Datensammlungen erlaubt. Die Schnittstelle *Iter* stellt eine Anzahl generischer Funktionen zur Iterationsabstraktion zur Verfügung. Die Funktionen dieser Schnittstelle ermöglichen unter anderem die Definition einer Anfragesprache für beliebige Bulk-Typen.

Die Funktionalität von Iteratoren umfaßt die Operationen der relationalen Algebra und des relationalen Kalküls. Durch Funktionen höherer Ordnung lassen sich Selektion und Projektion flexibel bestimmen, so daß auch SQL-ähnliche Anfragen möglich sind. Eine

ausführliche Einführung wird in [Niederee 92] gegeben. Die Schnittstelle *Sql* ermöglicht die Ausführung von SQL-Befehlen aus Tycoon-Anwendungen. Dabei wird die Funktonalität eines SQL-Servers durch polymorphe Funktionen repräsentiert.

Diese Schnittstelle erfüllt folgende Anforderungen:

- Transfer von variablen Daten zur Datenbank
- Transfer mengenwertiger Daten
- Repräsentation von NULL-Werten
- Fehlererkennung

Auch beim Zugriff auf SQL-Datenbanken können Iterationsabstraktionen der Schnittstelle *Iter* bei der Weiterverarbeitung von Anfrageergebnissen eingesetzt werden. Beispielsweise können einzelne Felder einer Tabelle visualisiert werden.

Folglich sind die Möglichkeiten zur Konstruktion und zum Umgang mit neuen Datatenstrukturen allein durch die Eigenschaften der Sprache TL bestimmt.

6.2.7 Transaktionen

Tycoon bietet zwei Ansätze zur transaktionalen Programmierung. Unter Ausnutzung von Funktionen höherer Ordnung und erster Klasse ist das Modul *transaction* der Bibliothek *dbenv* im *add-On* Ansatz implementiert [Matthes 93]. Es erlaubt neben flachen Transaktionen mit einer Undo-Log gesteuerten Fehlererholung auch das Setzen von Sicherungspunkten und die Verschachtelung umfangreicher Transaktionen [Niederee 92].

In einem alternativen Ansatz wird Tycoon mit einem für Workflows geeigneten Transaktionskonzept ausgestattet, welches auch extern verwaltete Daten mit einbezieht [Kornacker 95]. Das Modul persistent bietet Funktionen an, mit deren Hilfe flache Transaktionen mit persistenten Sicherungspunkten gesteuert werden können. Diese Schnittstelle stellt Funktionen zum Beenden und Sichern einer flachen Transaktion sowie zum Zurücksetzen einer Transaktion zur Verfügung. Eine langlebige Transaktion wird dabei in sequentielle Teilabschnitte partitioniert, die durch die Sicherungspunkte begrenzt werden. Die Persistenz der Sicherungspunkte garantiert nach einem Ausfall, daß der letzte durch einen Sicherungspunkt markierte Zustand wiederhergestellt werden kann.

6.2.8 Programmierung grafischer Benutzerschnittstellen

Die Unterstützung für die Entwicklung grafischer Benutzerschnittstellen in ist über einen *add-on* Ansatz in die Tycoon-Umgebung integriert. Es existieren zwei Schnittstellen, die die Integration externer Dienste zur Entwicklung grafischer Benutzerschnittstellen ermöglichen (Abbildung 6.4):

- Schnittstelle vom Tycoon-System zum NeWS-Toolkit [Müßig 94] und
- generische Schnittstelle zu StarView [Geisler 95].

Bei der Auswahl dieser externen Dienste zur Entwicklung grafischer Benutzerschnittstellen stehen die Aspekte der Portabilität und Objektorientierung im Vordergrund. Die grundlegenden Ziele der objektorientierten Programmierung, d.h. Vererbung, Wiederverwendung und Kapselung decken sich mit den Anforderung von Werkzeugen zur Benutzerschnittstellenerstellung. In folgenden werden Aspekte dieser Werkzeuge näher betrachtet.

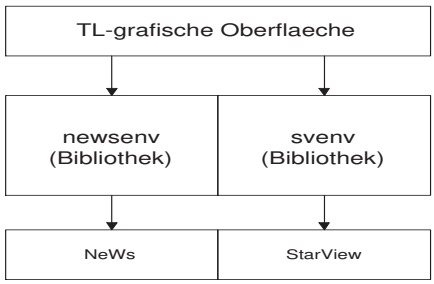

Abbildung 6.4: Entwicklung grafischer Benutzerschnittstelle in Tycoon

Entwicklung grafischer Benutzeroberflächen mit StarView

Ein generische Gateway-Generator ermöglicht die Integration externer Dienste (StarView) zur GUI-Programmierung in die persistente Tycoon-Umgebung [Geisler 95].

Starview ist eine portable C++ Klassenbibliothek zur GUI-Programmierung, die die Gestaltung und Entwicklung grafischer Benutzeroberflächen unabhängig vom jeweiligen Fenstersystem ermöglicht. Diese Bibliothek umfaßt mehr als 100 Klassen mit mehr als 500 Funktionen. Mit Hilfe dieser Klassen und dazugehörigen Methoden können ergonomische Benutzeroberflächen dargestellt werden.

In [Hachmann 96] ist eine vollständige Anleitung zur Oberflächenprogrammierung mit Tycoon unter Verwendung der StarView C++ Klassenbibliothek dargestellt. Ausgangspunkt der Entwicklung grafischer Benutzeroberflächen mit StarView ist die Erstellung der Fenster und die Festlegung dazugehöriger Interaktionselemente. Die TL-Anwendungen können die Klassen und Funktionen der StarView-Bibliothek (svenv) zur Festlegung der statischen Struktur der grafischen Interaktionselemente direkt aufrufen. Darüber hinaus ist eine interaktive Erstellung des Aussehens der Oberfläche mit dem Design Editor von StarView möglich. Nach dem Erzeugen und Initialisieren der Fenster werden die Ereignisse verarbeitet. Jedes Ereignis veranlaßt StarView eine benutzerdefinierte Funktion aufzurufen. Die ereignisbearbeitenden Funktionen können in TL geschrieben werden. Erfolgt ein Ereignis, so ruft StarView über *Callback* die ereigniszugehörige TL-Funktion auf und gibt somit die Steuerung der grafischen Benutzerschnittstelle an Tycoon weiter. Dieser Ablauf ist in der Abbildung 6.5 dargestellt.

Da die Entwicklung der StarView-Bibliothek und des Tycoon-Systems getrennt erfolgt, sind sie nicht auf der selben Anzahl von Plattformen vorhanden. Deswegen sind nicht alle Funktionen von StarView in TL-Anwendungsprogrammen verwendbar.

Neben der expliziten Programmierung der grafischen Benutzeroberflächen unterstützt die Schnittstelle zu StarView ein schnelles Entwerfen von Fenstern mit einheitlichen Bedienelementen und Funktionen zu einer direkten Manipulation von beliebigen strukturierten Objekten (Record, Tupel, Listen, usw.)

Eine komplexe grafische Benutzerschnittstelle läßt sich nicht nur aus Kombination dieser Fenster realisieren. Diese Fenster können auch sehr gut zur Prototyping eingesetzt werden. Bei der Realisierung dieser Fenster werden die Funktionen der StarView Bibliothek (svenv) über eine Zwischenschicht (sveditenv) angesprochen. Damit wird die

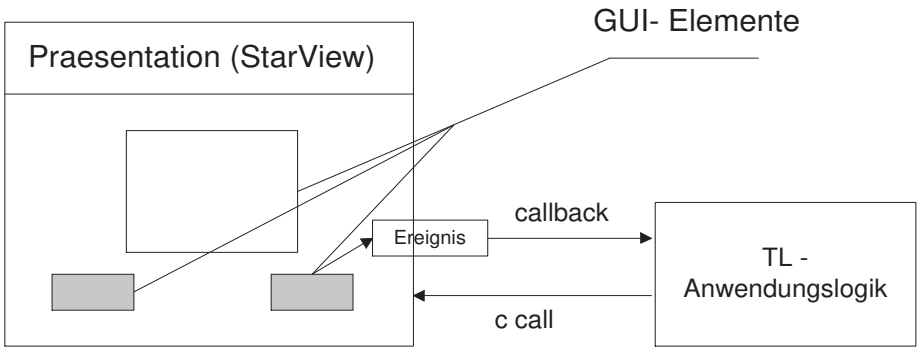

Abbildung 6.5: Aufruf von TL-Funktionen aus StarView-Präsentation

Unabhängigkeit der Anwendung von Änderungen der Implementation von einzelnen Funktionen erreicht.

Entwicklung grafischer Benutzeroberflächen mit NeWS-Toolkit

Das NeWS-Toolkit [SunSoft 92] ist ein auf der Sprache PostScript [2] [Adobe Systems 91] basierendes objektorientiertes Werkzeug zur Erstellung von Benutzerschnittstellen für das Fenstersystem NeWS [3]. Das Werkzeug implementiert viele der OPEN LOOK-Schnittstellenkomponenten für grafische Benutzerschnittstellen. C-basierte Bibliotheken zur Kommunikation zwischen dem Anwendungsprogramm und dem Fenstersystem sowie zur Ereignisverwaltung ermöglichen den Einsatz des Werkzeuges unter Verwendung verschiedener Programmiersprachen.Basierend auf diese Eigenschaft bietet die Tycoon-Bibliothek *newsenv* eine Schnittstelle zum NeWS-Toolkit, die die Klassen des NeWS-Toolkit in Tycoon verfügbar macht [Müßig 94].

Zur Erstellung von Benutzerschnittstellen sind im NeWS-Toolkit über 30 Klassen vordefiniert, die die Darstellung von Fenstern und von GUI-Elementen, ermöglichen. Die Anwendungsentwicklung mit dem NeWS-Toolkit besteht aus zwei Teilen. Der Server-Teil, d.h. die Festlegung das Aussehen der Benutzerschnittstelle, wird in der objektorientierten NeWS-Sprache geschrieben. Der Client-Teil, d.h. das Anwendungsprogramm, kann z.B. in C oder C++ geschrieben werden. Zu jeder Klasse des NeWS- Toolkits gibt es in der *newsenv* Bibilothek ein korrespondierndes Modul. Unter Benutzung dieser Module können beliebige grafische Benutzerschnittstellen für Tycoon-Programme erzeugt werden.

Basierend auf dieser Schnittstelle ist die Bibliothek *editenv* implementiert, die die notwendigen Dienste für eine typsichere generische Datenvisualisierung in Tycoon bietet.

6.3 Entwicklungswerkzeuge

Im Vergleich zu kommerziellen Entwicklungsumgebungen, die eine große Anzahl von Entwicklungswerkzeuge anbieten, sind die Entwicklungswerkzeuge im Tycoon-System recht überschaubar. Alle Tätigkeiten im Tycoon-System (Übersetzen, Binden, Ausführen, Feh-

[2] Die Sprache ist eine von Adobe System Inc. entwickelte, dynamische, kellerspeicherbasierte und interpretative Seitenbeschreibungssprache.

[3] NeWS ist ein von Sun Microsystems ab 1984 entwickltes netzwerkfähiges Fenstersystem

leranalyse, Definition von Systemparametern) werden durch Kommandos (*top level phrases*) im interaktiven Tycoon *top level* ausgelöst.

Auch bei der Auswahl von Werkzeugen folgt in Tycoon der *add-on* Ansatz. Das Tycoon-System stellt den sprachlichen und architektonischen Rahmen für eine flexible Integration externer Dienste bereit. Neben der im Abschnitt 6.4.2 vorgestellten Präsentationsdienste (StarView, News) und Systemdienste(C/C++) werden auch Modellierungswerkzeuge (Stp/OMT) und Werkzeuge zur Entwicklung grafischer Oberflächen (Design Editor, Devguide) in das System integriert. Für die Abwicklung von Softwareprojekten im Team werden die Dienste der Teamware-Umgebung, ein Bestandteil der Softwareentwicklungsumgebung von Sun Microsystem, benutzt.

In diesem Abschnitt werden die in Tycoon integrierten, externe Werkzeuge kurz vorgestellt.

Werkzeuge zur Modellierung auf Basis der OMT [4]

In [Nietsch 96] wird eine Schnittstelle zur Case-Umgebung StP-OMT entwickelt. Neben einer Benutzeroberfläche, die verschiedene Werkzeuge zur Modellierung auf Basis der OMT integriert, umfaßt Stp-OMT das *Object Management System (OMS)*, das *persistent Data Model (PDM)* und ein Repository zur persistenten Datenspeicherung.

Werkzeuge zur Entwicklung grafischer Oberflächen

Die Schnittstellen von Tycoon-System zum NeWS Toolkit und StarView (s. Abschnitt 6.2.8) ermöglichen auch mit dem Devguide und Design-Editor erzeugte grfaische Benutzerschnittstellen in einer Tycoon-Anwendungen anzubinden:

- Design Editor: Zur Gestaltung der grafischen Benutzerschnittstelle bietet StarView den Design Editor. Mit dem Design Editor kann die statische Struktur der grafischen Interaktionselemente (Menüs, Dialog, Arbeitsfenster etc.) sehr einfach erstellt werden. Die Eigenschaften dieser Elemente können ebenfalls interaktive festgelegt werden. Eine komplette Anwendung kann nicht mit dem Design Editor erstellt werden. Es existiert keine grafische Umgebung für den Entwurf und die Realisierung von Aktionen auf Ereignisse. Diese müssen explizit programmiert werden.

- Devguide: Mit Hilfe des Devguides (*Developers Graphical User Interface Design Editor*) [Sun Microsystems 92] ist es möglich, interaktiv die Benutzerschnittstelle zu erstellen. Schon während der Entwicklung mit dem Devguide kann das endgültige Aussehen und Verhalten der Benutzerschnittstelle in einem Test-Modus simuliert werden. Umfangreiche Dialogmanagementfunktionen ermöglichen die Ansteuerung vieler Interaktionselemente der Benutzerschnittstelle, ohne daß dieses Verhalten explizit codiert werden muß.

Workgroup-Funktionalität

Die Arbeit mehrerer Entwickler an eine Software wird durch Dienste der Teamware-Umgebung unterstützt. Sie ist ein Bestandteil von SCCS, die sprachunabhängige Softwareentwicklungsumgebung von Sun Microsystem. Zur Koordination und Verwaltung der Arbeit im Team bietet dieses Werkzeug folgende Komponente:

[4] Ein Programmierstil der häufig methodenorientiert genannt wird, im Vergleich zu Objektorientierung, die allerdings weitere Spracheigenschaften, wie Vererbung und Kapselung voraussetz.

- VersionTool: Die Versionen von einem Quelltext kann mit diesem grafisch orientierten Werkzeug verwaltet werden.
- FilteMerger: Dieses Werkzeug ermöglicht den Entwickler zwei unterschiedliche (aber meist von einer Datei abstammende) Quelldateien zu einer einzigen zu verschmelzen.
- Codemanager: Der Codemanager ist die Komponente, welche die Aktivitäten der an einem Projekt beteiligten Entwickler koordiniert.
- ParallelMake: Das ParallelMake bietet ein grafisches Frontend zum mak-Programm.

Eine ausführliche Beschreibung über die Komponenten und Konzepte von SCCS ist in [Koch 97] gegeben.

Softwaremanagement-Werkzeuge

Zum Einsatz von Software in verteilten Umgebungen stehen die Bibliothekdienste zur Verfügung, die in der interaktiven Tycoon *top level* Umgebung abrufbar sind.

- Die Dienste des Moduls *port* ermöglichen das plattformunabhängige Übertragen von Tycoon-Objekte beliebiger Typs zwischen Tycoon-Anwendungen im Netz.
- Automatische Installation und Deinstallation von Objekten: Mechanismen zur automatischen Objektreplikation ([Mathiske 96], [Vaziri Pour 96]) ermöglichen eine flexible Installation und Deinstallation von Software in einer verteilten Umgebung. Automatische Lastverteilung: Migrierende, persistente Threads können als Konstrukte für die explizite Lastverteilung eingesetzt werden. Bei der hohen Belastung eines Servers migrieren wartende Threads auf einen Server, der weniger belastet ist. Die Migration von Diensterbringer auf anderen Server wird von der dynamischen, hierarchisen Adressierung [Göllnitz 96] unterstützt.

Kapitel 7

Zusammenfassung und Ausblick

Nach einer umfangreiche Studie der Informatik-Fachliteratur wurde in dieser Arbeit ein Kriterienkatalog mit mehr als 100 technischen Kriterien entwickelt, die interne Qualitätsmerkmale einer Entwicklungsumgebung darstellen. In diesem Katalog werden interne Qualitätsmerkmale definiert und ihre Relevanz bei der Entwicklung unternehmensweiter Client/Server-Anwendungen zum Erreichen externer Qualitätsmerkmale diskutiert. Ziel dieser Katalog ist es, die gesammelten Informationen als Hilfestellung für Anwender bei der Auswahl einer Software zur Entwicklung von Client/Server-Anwendungen zu bieten.

Es ist natürlich im Rahmen einer Diplomarbeit nicht möglich, einen vollständigen und erschöpfenden Kriterienkatalog aufzustellen. Der in dieser Arbeit entwickelte Katalog erfaßt jedoch die drei Aspekte von Client/Server-Entwicklungsumgebungen (Sprache, Architektur und Entwicklungswerkzeuge) in erheblicher Detaillierung. Diese Aspekte sind in einem iterativen Vorgang aus Konzepten der Softwaretechnik und als Ergebnis der Untersuchung von drei Systeme NATURAL, ABABP/4 und Tycoon abgeleitet worden. Es wurden bewußt drei Systeme unterschiedlicher Leistungsfähigkeiten und Zielsetzung ausgewählt, um sowohl praxisbezogene als auch neue Innovationen berücksichtigende Kriterien zu erhalten.

Damit kann dieser Katalog bei einem Entscheidungsprozeß zur Auswahl einer Client/Server-Entwicklungsumgebung verwendet werden. Als Begleitsmaterial zu diesem Katalog sind computergestützte Hilfsstrukturen in Form von Excel-Tabellen (s. Anhang A) erstellt worden. Die im Katalog erfaßten internen Qualitätsmerkmale sind als Fragen in Ergebnistabellen formuliert.

Diese Ergebnistabellen bilden zusammen mit den Korrelationstabellen (s. Anhang B), die den Bezug zwischen den internen Qualitätsmerkmalen und den externen Qualitätsmerkmalen (den Anforderungen an Client/Server-Anwendungen) darstellen, den Kern der Ergebnisse dieser Arbeit.

7.1 Bewertungsmöglichkeiten

Ziel dieser Arbeit ist die Bereitstellung strukturierter Kriterien, die einem Entscheider Basiskenntnisse und eine Methodik für die systematische Untersuchung und die Evaluation von Client/Server-Entwicklungsumgebungen bieten. Es wird dabei jedoch keine Entscheidung für den Benutzer getroffen. Bei der Auswahl einer Software für ein konkretes Unternehmen werden typischerweise weitere firmenspezifische Faktoren eine entscheidende Rolle spielen. Beispiele für diese Faktoren sind die Komplexität der Anwendung, die An-

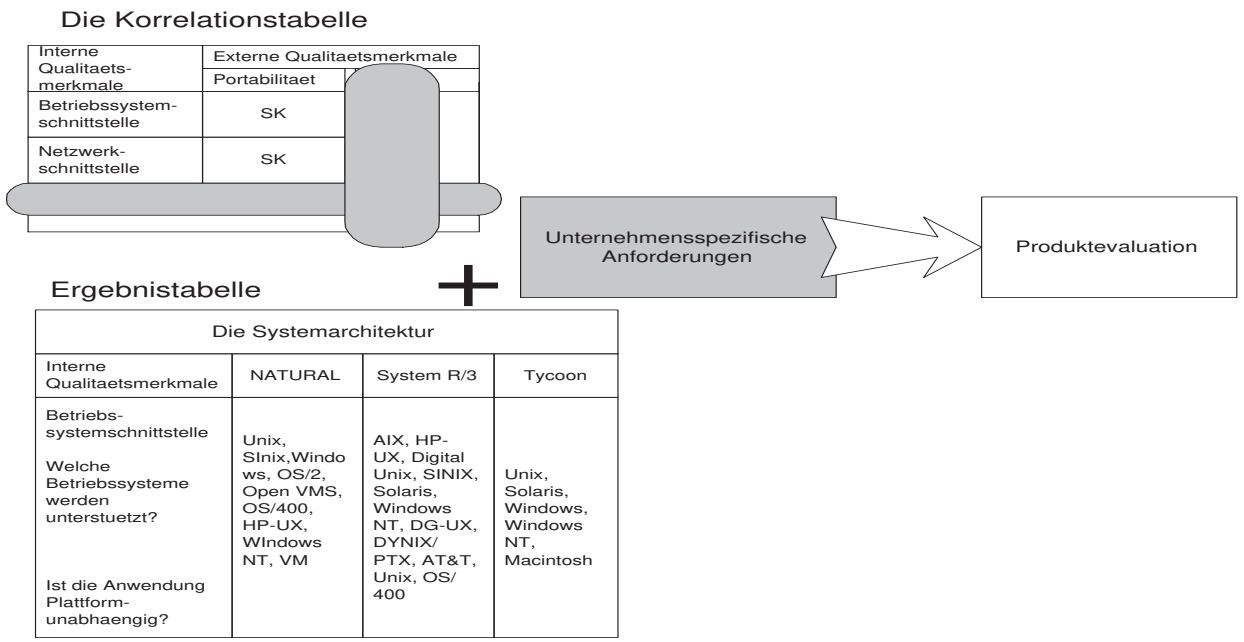

Abbildung 7.1: Produktevaluation

zahl der Benutzer, die im Unternehmen vorhandenen softwaretechnischen Erfahrungen, die bevorzugte Entwurfsmethodik usw. Diese Faktoren können dann, wie in Abbildung 7.1 dargestellt ist, als Gewichtungsfaktoren in Ergebnis-Tabellen eingeführt werden und ermöglichen eine Bewertung der Software im Licht der speziellen zu realisierenden Firmeninteressen.

Eine alternative Bewertungsmöglichkeit ist zu bevorzugen, wenn ein bestimmtes externes Qualitätsmerkmal, z.B. Interoperabilität, stark bei der Auswahl einer Software im Vordergrund steht. In diesem Fall können, wie in Abbildung 7.2, die mit Interoperabilität positiv korrelierten internen Qualitätsmerkmale in einer Tabelle gruppiert werden. Diese Ergebnistabellen bieten dann überschaubare Softwareinformationen bzgl. des ausgewählten externen Qualitätsmerkmals.

7.2 Nutzung der Ergebnisse dieser Arbeit

Mit NATURAL, System R/3 und Tycoon stehen sich drei Systeme unterschiedlicher Leistungsfähigkeit und Zielsetzung gegenüber. NATURAL und System R/3 setzen ihren Schwerpunkt deutlich im betriebswirtschaftlichen Bereich und benutzen historisch gewachsene und wirtschaftlich bedingte System- und Modellierungstechniken. Tycoon hingegen implementiert moderne Konzepte, besitzt aber noch keine betriebswirtschaftlichen Bibliotheken oder Anbindungen an entsprechende Systeme. Daher wird in dieser Arbeit keine direkte Gegenüberstellung dieser drei Systemen gestrebt, sondern es werden die internen Qualitätsmerkmale dieser Systeme bzgl. der externen Qualitätsmerkmale der Client/Server-Anwendungen untersucht. Damit soll nur die prinzipielle Einsetzbarkeit dieser Systeme für die Entwicklung unternehmensweiter Client/Server-Anwendungen demonstriert werden.

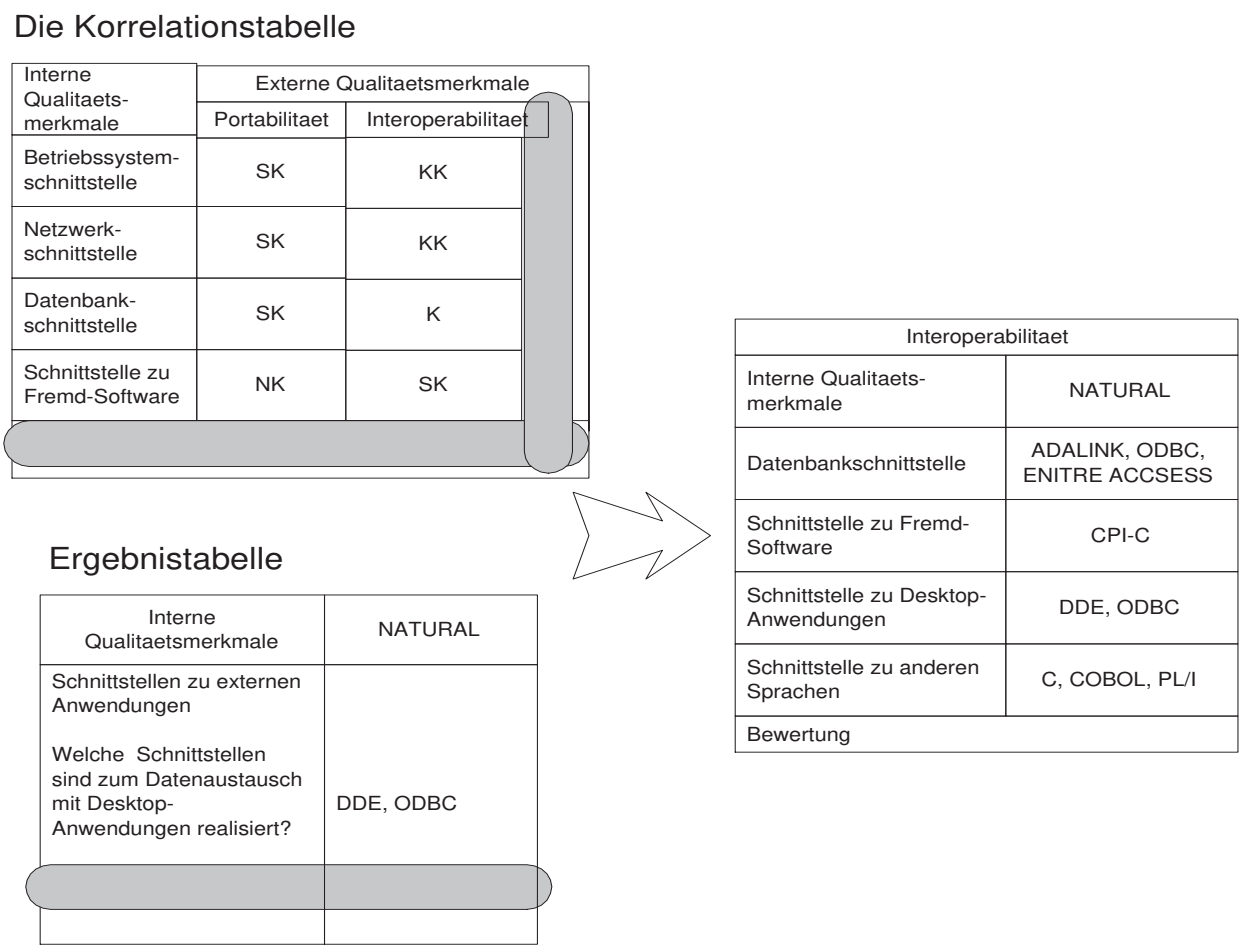

Abbildung 7.2: Produktevaluation anhand der mit Interoperabilität positiv korrelierten, internen Qualitätsmerkmale

Die Ergebnisse dieser Untersuchung und der Gegenüberstellung der drei Systeme sind sowohl für die Forschung und Lehre als auch für die Kooperation zwischen Universität und Industrie zwecks Technologietransfer von Bedeutung. Denkbar sind zunächst folgende Einsatzmöglichkeiten in der Forschung und Lehre:

- Wegweisung für die weitere werkzeugunterstützte Client/Server-Entwicklung in Tycoon,
- Analyse der Stärken und Schwächen existierender Client/Server-Technolgie und
- Darstellung der Integrationsmöglichkeiten der Dienste dieser Systemen in einer Client/Server-Umgebung.

Im Rahmen des Technologietransfers kann die praxisorientierte Darstellung der Client/Server- Aspekte des Tycoon-Systems für Systementwickler von NATURAL oder ABAP/4 von Bedeutung sein. Darüber hinaus trägt die Einordnung der Aspekte von NATURAL in die Client/Server-Referenzarchitektur zur Standortbestimmung auf dem Weg vom Mainframe zur Client/Server-Architektur bei.

7.3 Ausblick

Der Kriterienkatalog ist zum großen Teil aus der Literatur extrahiert, daher kann er nicht nur für die in dieser Arbeit untersuchten Systeme verwendet werden, sondern er kann auch zur Evaluation und Gegenüberstellung anderer Entwicklungsumgebungen z.B. Java, Tycoon2 oder Borland Delphi eingesetzt werden. Die Struktur des Kriterienkatalogs erlaubt seine Erweiterung bzw. Modifikation. Die hierarchisch strukturierten internen Qualitätsmerkmale können um neue Konzepte z.B. Objektorientierung erweitert werden. Der Umfang des Kriterienkatalog kann auch entsprechend durch weitere Verfeinerung von Kriterien oder durch Abstraktion von Teilkriterien geändert werden.

Die benutzte Methodik bietet sich auch für den Vergleich von Entwicklungsumgebungen mit anderen Schwerpunkten an. Beispielsweise bilden bei der Evaluation von Entwicklungswerkzeugen für Workflow-Anwendungen Anforderungen an Workflow-Anwendungen zusätzliche externe Qualitätsmerkmale. Die internen Qualitätsmerkmale werden in diesem Fall um neue Konzepte, wie zum Beispiel Modifizierbarkeit von Ablaufspezifikationen durch den Benutzer erweitert. Wichtig ist dabei die Diskussion der Relevanz von internen Qualitätsmerkmalen bzgl. der neuen externen Qualitätsmerkmale, die später auch bei der Bewertung zur Geltung kommen.

Literaturverzeichnis

Adobe Systems 91: Adobe Systems, Inc. (Hrsg.). *PostScript Language. Tutorial and Cookbook.* Addison-Wesley Publishing Company, 1991.

Atkinson, Bunemann 87: Atkinson, M.P. und Bunemann, P. *Types and Persistence in Database Programming Languages.* ACM Computing Surveys, Jg. 19, Juni 1987, Nr. 2.

Böhm, Jacopini 66: Böhm, D. und Jacopini, G. *Flow Diagrams, Turing Machines and Languages with only two Formation Rules.* Communications of the ACM, Jg. 9, 1966.

Borgida et al. 93: Borgida, A., Mylopoulos, J., und Schmidt, J. *The TaxisDL Software Description Language.* In: *Database Application Engineering with DAIDA.* Springer-Verlag, 1993, S. 65–84.

Buck-Emden et al. 93: Buck-Emden, R., Hansen, W.R., Meyer, H.M., Mendoza, E., und Jacobi, A.G. *Grundlagen für den Entwurf innovativer betriebswirtschaftlicher Anwendungs- und Informationssysteme.* In: *Client-Server/Architektur, Grundlagen und Herstellerkonzepte für Downsizing.* elsevier, 1993.

Buck-Emden, Galimow 95: Buck-Emden, R. und Galimow, J. *Die Client/Server-Technologie des SAP-Systems R/3.* Addison-Wesley Bonn, 1995.

Cardelli et al. 88: Cardelli, L., Donahue, J., Glassman, L., Jordan, M., Kalsow, B., und Nelson, G. *Modula-3 Report.* Technical Report ORC-1, Olivetti Research Center, 2882 Sand Hill Road, Menlo Park, California, 1988.

Cardelli, Wegner 85: Cardelli, L. und Wegner, P. *On Understanding Types, Data Abstraction and Polymorphism.* ACM Computing Surveys, 1985.

Cardelli 97: Cardelli, L. *Global Computing.* Digital Equipment Coroperation, 1997.

Date 89: Date, C. J. *A Guide to the SQL Standard.* Reading, 1989.

Denert 91: Denert, F. *Software-Engineering.* Springer-Verlag, 1991.

Forge 94: Forge, S. *Developing cooperative and client server Systems, A Manager's Guide.* McGraw-Hill Book Co. Europa, 1994.

Geisler 95: Geisler, A. *Basisdienste zur Gestaltung einer reflektiven grafischen Entwicklungsumgebung für eine persistente Programmiersprache.* Diplomarbeit, Fachbereich Informatik, Universität Hamburg, 1995.

Göllnitz 96: Göllnitz, M. *Polymorphe, persistente Client/Server-Programmierung mit dynamischer, hierarchischer Adreßauflösung.* Studienarbeit, Fachbereich Informatik, Universität Hamburg, 1996.

Goos, Kastens : Goos, G. und Kastens, U. *Programming languages and the design of modular programs.*

Gorzynski 95: Gorzynski, L. *Entfernter Datenzugriff auf persistente Objektspeicher unter Benutzung standardisierter Protokolle (TSP/NFS.* Studienarbeit, Fachbereich Informatik, Universität Hamburg, 1995.

Hachmann 96: Hachmann, I. *Generische Visualisierungsdienste am Beispiel von StarView und Tycoon.* Studienarbeit, Fachbereich Informatik, Universität Hamburg, 1996.

Hansen et al. 93: Hansen, W.R., Meyer, H.M., Buck-Emden, R., Mendoza, E., und Jacobi, A. G. *Client-Server/Architektur, Grundlagen und Herstellerkonzepte für Downsizing.* In: *Client-Server/Architektur, Grundlagen und Herstellerkonzepte für Downsizing.* elsevier, 1993.

Heinrich, Roithmay 89: Heinrich, L.J. und Roithmay, F. *Wirtschaftsinformatik-Lexikon.* 1989.

Hoare 68: Hoare, C.A.R. *Record Handling.* In: Genuys, F. (Hrsg.). *Programming Languages.* Academic Press, New York and London, 1968, S. 291–347.

Johannisson 95: Johannisson, N. *Generische Programmierung typsicherer Kommunikationsdienste.* Studienarbeit, Fachbereich Informatik, Universität Hamburg, 1995.

Keppel 93: Keppel, D. *Tools and Techniques for Building Fast Portable Threads Packages.* Technical Report 93-05-06, University of Washington, Department of Computer Science and Engineering, University of Washington, Seattle, Mai 1993.

Knolmeyer 92: Knolmeyer, G. *Downsizing in Wirtschaftsinformatik.* 1992.

Koch 97: Koch, Ch. *Entwicklungsunterstützung für persistente strukturierte Systeme.* Diplomarbeit, Fachbereich Informatik, Universität Hamburg, 1997.

Kornacker 93: Kornacker, M. *Eine polymorphe typisierte SQL-Schnittstelle.* Studienarbeit, Fachbereich Informatik Universität Hamburg, 1993.

Kornacker 95: Kornacker, M. *Persistente Sicherungspunkte für langlebige Aktivitäten in offenen Umgebungen.* Diplomarbeit, Fachbereich Informatik Universität Hamburg, 1995.

Lutz 97: Lutz, S. *Eine polymorph typisierte Schnittstelle der Sprache Tycoon zum System SAP R/3.* Diplomarbeit, Fachbereich Informatik Universität Hamburg, 1997.

Mathiske et al. 95: Mathiske, B., Matthes, F., und Schmidt, J.W. *On Migrating Threads.* In: *Proceedings of the Second International Workshop on Next Generation Information Technologies and Systems, Naharia, Israel,* Juni 1995. (Also appeared as TR FIDE/95/136).

Mathiske 92: Mathiske, B. *Kodegenerierung für Programmiersprachen mit Persistenz, Polymorphie und Funktionen höherer Ordnung.* Diplomarbeit, Fachbereich Informatik Universität Hamburg, 1992.

Mathiske 96: Mathiske, B. *Mobilität in persistenten Objektsystemen.* Dissertation, Fachbereich Informatik Universität Hamburg, 1996.

Matthes et al. 94: Matthes, F., Müßig, S., und Schmidt, J.W. *Persistent Polymorphic Programming in Tycoon: An Introduction.* FIDE Technical Report FIDE/94/106, Fachbereich Informatik, Universität Hamburg, Germany, August 1994.

Matthes et al. 95: Matthes, F., Müller, R., und Schmidt, J.W. *Towards a Unified Model of Untyped Object Stores: Experience with the Tycoon Store Protocol.* In: Atkinson, M.P. (Hrsg.). *Fully Integrated Data Environments.* Springer-Verlag (to appear), 1995.

Matthes, Schmidt 94: Matthes, F. und Schmidt, J.W. *Persistent Threads.* In: *Proceedings of the Twentieth International Conference on Very Large Data Bases, VLDB*, September 1994.

Matthes, Schmidt 97: Matthes, F. und Schmidt, J.W. *Datenbankmodelle und Datenbanksprachen.* Springer-Verlag, 1997.

Matthes 93: Matthes, F. *Persistente Objektsysteme: Integrierte Datenbankentwicklung und Programmerstellung.* Springer-Verlag, 1993.

Matzke 96: Matzke, B. *Die Programmiersprache des SAP-Systems R/3.* Addison-Wesley, 1996.

Meyer 90: Meyer, B. *Objektorientierte Softwareentwicklung.* Prentic Hall, 1990.

Microsoft 92: Microsoft. *Microsoft ODBC Application Programmers Guide.* Document, 1992.

Mühlhäuser 86: Mühlhäuser, M. *Entwicklungsunterstützung für anwendungsorientierte verteilte Programme.* Dissertation, Fakultät für Informatik Universität Karlsruhe, 1986.

Müßig 94: Müßig, S. *Beiträge zur typsicheren generischen Datenvisualisierung.* Diplomarbeit, Fachbereich Informatik, Universität Hamburg, 1994.

Niederee 92: Niederee, C. *Generische Dienste für datenintensive Anwendungen: Iterationsabstraktion, Integritätsüberwachung Fehlererholung.* Diplomarbeit, Fachbereich Informatik, Universität Hamburg, 1992.

Niemann 95: Niemann, K.D. *Client/Server-Architektur, Organisation und Methodik der Anwendungsentwicklung.* Vieweg, 1995.

Nietsch 96: Nietsch, O. *Die Anbindung des Tycoon-Systems an die Case-Umgebung STP über ein Repository.* Studienarbeit, Fachbereich Informatik, Universität Hamburg, 1996.

Orfali et al. 96: Orfali, R., Harkey, d., und J., Edwards. *The Essential Client/Server Survival Guide.* John Whiley & Sons, inc., 1996.

Potter, Dr. Cory 93: Potter, C. und Dr. Cory, Th. *Client/Server Development Tools, An Evaluation and Comprarison.* ButlerBloor Limited, 1993.

Rosenhagen 95: Rosenhagen, J. *Client/Server statt Codd und Date.* Client/Server Computing, Jg. 2, 1995.

Scheu 95: Scheu, S. *Effiziente NATURAL-Programmierung.* Vieweg, 1995.

Schmitt 93: Schmitt, H.J. *Client/Server Architekturen, Architekturmodelle für eine neue Informationstechnische Infrastruktur.* PETER LANG, 1993.

Scotts 82: Scotts, P.D.Jr. *A Comparative Survey of Concurrent Programming Languages.* ACM SIGPLAN Notices, 1982, Nr. 10.

Seibt 92: Seibt, D. *Individuelle Datenverarbeitung.* Handwörterbuch der Organisation, 1992.

Smith 77: Smith, J.M. *Database Abstractions: Aggregation and Generalization.* ACM Transactions on Database Systems, Jg. 2, Juni 1977, Nr. 2.

Software 90: Software, AG. *NATURAL Konzept und Funktionalität.* Document, 1990.

Software 93: Software, AG. *NATURAL 2.2 Leitfaden zur Programmierung.* Document, 1993.

Software 96a: Software, AG. *NATURAL LightStorm 3.2, Application Engineering.* Electronic Document NLS3211296CDD, 1996.

Software 96b: Software, AG. *NATURAL LightStorm 3.2, Concepts and Facilities & Getting Started.* Document NLS321-006WIN, 1996.

Software 96c: Software, AG. *NATURAL LightStorm 3.2, Engineering Workbench Methods Guide.* Document NLS321-032WIN, 1996.

Steffens 96: Steffens, U. *Integration von Information Retrieval Funktionalität in eine offene, persistente Programmierumgebung.* Studienarbeit, Fachbereich Informatik, Universität Hamburg, 1996.

Sun Microsystems 92: Sun Microsystems, Inc. *OpenWindows Developers Guide 2.0 Users Guide.* Mountain View, 1992.

SunSoft 92: SunSoft, Inc. *The NeWS Toolkit 3.1 Reference Manual.* Mountain View, 1992.

Swenson 93: Swenson, K.D. *Visual Support for Reengineering Work Processes.* In: *Proceedings of the Conference on Organizational Computing Systems, COOCS'93.* ACM Press, 1993.

Turner 93: Turner, K. *Using Formal Description Techniques, An Introduction to Estelle, Lotos and SDL.* John Wiley & Sons, 1993.

Vaziri Pour 96: Vaziri Pour, N. *Flexible Bindungstechniken für ubiquitäre Ressourcen in verteilten Anwendungen.* Studienarbeit, Fachbereich Informatik, Universität Hamburg, 1996.

Walker, Popek 89: Walker, B.J. und Popek, G.J. *A Transparent Environment.* BYTE, 1989, Nr. 14.

Watt 90: Watt, A. *Programming Language Concepts und Paradigms.* Prentic Hall, 1990.

Erklärung

Ich erkläre hiermit, die vorliegende Arbeit selbständig durchgeführt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben.

Hamburg, 12. September 1997

Nastaran Vaziri Pour